

Oracle® Application Server

Developer's Guide: PL/SQL and ODBC Applications

Release 4.0.8.1

September 1999

Part No. A66958-02

ORACLE®

The programs are not intended for use in any nuclear, aviation, mass transit, medical, or other inherently dangerous applications. It shall be the licensee's responsibility to take all appropriate fail-safe, backup, redundancy, and other measures to ensure the safe use of such applications if the programs are used for such purposes, and Oracle Corporation disclaims liability for any damages caused by such use of the programs.

The programs (which include both the software and documentation) contain proprietary information of Oracle Corporation; they are provided under a license agreement containing restrictions on use and disclosure and are also protected by copyright, patent, and other intellectual and industrial property laws. Reverse engineering, disassembly, or decompilation of the programs is prohibited.

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. Oracle Corporation does not warrant that this document is error free. Except as may be expressly permitted in your license agreement for these programs, no part of these programs may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Oracle Corporation.

If the programs are delivered to the U.S. Government or anyone licensing or using the programs on behalf of the U.S. Government, the following notice is applicable:

Restricted Rights Notice Programs delivered subject to the DOD FAR Supplement are "commercial computer software" and use, duplication, and disclosure of the programs, including documentation, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement. Otherwise, programs delivered subject to the Federal Acquisition Regulations are "restricted computer software" and use, duplication, and disclosure of the programs shall be subject to the restrictions in FAR 52.227-19, Commercial Computer Software - Restricted Rights (June, 1987). Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065.

Oracle is a registered trademark, and the Oracle logo, NLS*WorkBench, Pro*COBOL, Pro*FORTRAN, Pro*Pascal, SQL*Loader, SQL*Module, SQL*Net, SQL*Plus, Oracle7, Oracle Server, Oracle Server Manager, Oracle Call Interface, Oracle7 Enterprise Backup Utility, Oracle TRACE, Oracle WebServer, Oracle Web Application Server, Oracle Application Server, Oracle Network Manager, Secure Network Services, Oracle Parallel Server, Advanced Replication Option, Oracle Data Query, Cooperative Server Technology, Oracle Toolkit, Oracle MultiProtocol Interchange, Oracle Names, Oracle Book, Pro*C, and PL/SQL are trademarks or registered trademarks of Oracle Corporation. All other company or product names mentioned are used for identification purposes only and may be trademarks of their respective owners.

Contents

Preface	vii
----------------------	-----

Part I PL/SQL Cartridge

1 PL/SQL Cartridge Overview

Configuration Information	1-1
Communication Path	1-2
POST and GET Methods	1-4

2 Tutorial

1. Creating and Loading the Stored Procedure onto the Database	2-2
2. Creating an Application and its Components	2-3
3. Reloading	2-5
4. Creating an HTML Page to Invoke the Application	2-6

3 Adding and Invoking PL/SQL Applications

Adding PL/SQL Applications	3-1
Adding Applications and Cartridges	3-2
Creating Database Access Descriptors (DADs)	3-3
Starting and Stopping the Application Server	3-5
Adding Cartridges to an Existing Application	3-5
Configuring PL/SQL Applications	3-7
Application Configuration	3-7
Cartridge Configuration	3-7

Invoking a PL/SQL Cartridge	3-8
URL Format	3-8
Caching Procedure Descriptions	3-10
Overloaded Procedures	3-10
Overloading and PL/SQL Arrays	3-11
Variables with Multiple Values	3-12
Flexible Parameter Passing	3-16
Positional Parameters	3-17
Executing SQL Files	3-18
Location of PL/SQL Source Files	3-19
Parameters	3-19
Life Cycle of the PL/SQL Cartridge	3-20
Initialization	3-20
Authorization	3-20
Execution	3-20
Shutdown	3-21

4 Using the PL/SQL Web Toolkit

Common Schema	4-1
PL/SQL Web Toolkit Installation	4-2
Packages in the Toolkit	4-4
htp and htf	4-5
owa_image	4-6
owa_opt_lock	4-6
owa_custom	4-6
owa_content	4-7
Conventions for Parameter Names in the Toolkit	4-8
Attributes to HTML Tags	4-8
PL/SQL Cartridge and Applets	4-8
Sessions/Cookies	4-9
LONG Data Type	4-10
Inter-Cartridge Exchange (ICX)	4-10
Extensions to the htp and htf Packages	4-10
File Upload and Download	4-11
Upload	4-11

Download	4-14
String Matching and Manipulation	4-17
owa_pattern.match.....	4-17
owa_pattern.change	4-18
 5 Authentication and Security	
Dynamic Username/Password Authentication	5-1
Dynamic Username/Password and the Basic_Oracle Scheme	5-2
PL/SQL Cartridge and Authentication Server Schemes	5-2
Custom Authentication.....	5-4
OWA_SEC.GLOBAL.....	5-5
OWA_SEC.PER_PACKAGE.....	5-6
OWA_SEC.CUSTOM	5-7
 6 Transactions	
Mechanics of Transaction Service.....	6-1
Example.....	6-3
 7 Miscellaneous	
Supported Data Types.....	7-1
NLS Extensions	7-1
Upgrading PL/SQL Cartridge from 3.x to 4.0	7-3
 8 Troubleshooting	
Problems with Invoking Your PL/SQL Application	8-1
Looking at Error Messages Generated by the Database	8-1
Unhandled Exceptions.....	8-2
Looking at the HTML Generated by Your PL/SQL Application	8-2
Tracing Levels	8-4
Error-Reporting Levels	8-5
 Part II ODBC Cartridge	

9 ODBC Cartridge Overview

Review of Cartridge Architecture	9-1
Supported Data Sources	9-2

10 Using the ODBC Cartridge

Invoking the ODBC Cartridge	10-1
Special Usage Considerations	10-2

11 Specifying Modes and Datatypes

Specifying a Request Mode	11-1
Request Mode Parameters	11-1
Execute Mode	11-3
TablePrint Mode	11-3
StringPrint Mode	11-4
Specifying Datatypes	11-4

Index

Preface

Audience

This book is intended for people who develop Web applications using the PL/SQL and ODBC cartridges of Oracle Application Server 4.0.

The Oracle Application Server Documentation Set

This table lists the Oracle Application Server documentation set.

Title of Book	Part No.
Oracle Application Server 4.0.8 Documentation Set	A66971-03
Oracle Application Server Overview and Glossary	A60115-03
Oracle Application Server Installation Guide for Sun SPARC Solaris 2.x	A58755-03
Oracle Application Server Installation Guide for Windows NT	A58756-03
Oracle Application Server Administration Guide	A60172-03
Oracle Application Server Security Guide	A60116-03
Oracle Application Server Performance and Tuning Guide	A60120-03
Oracle Application Server Developer's Guide: PL/SQL and ODBC Applications	A66958-02
Oracle Application Server Developer's Guide: JServlet Applications	A73043-01
Oracle Application Server Developer's Guide: LiveHTML and Perl Applications	A66960-02
Oracle Application Server Developer's Guide: EJB, ECO/Java and CORBA Applications	A69966-01
Oracle Application Server Developer's Guide: C++ CORBA Applications	A70039-01
Oracle Application Server PL/SQL Web Toolkit Reference	A60123-03
Oracle Application Server PL/SQL Web Toolkit Quick Reference	A60119-03

Title of Book	Part No.
Oracle Application Server JServlet Toolkit Reference	A73045-01
Oracle Application Server JServlet Toolkit Quick Reference	A73044-01
Oracle Application Server Cartridge Management Framework	A58703-03
Oracle Application Server 4.0.8.1 Release Notes	A66106-04

Conventions

This table lists the typographical conventions used in this manual.

Convention	Example	Explanation
bold	oas.h owsctl wrbcfg www.oracle.com	Identifies file names, utilities, processes, and URLs
italics	<i>file1</i>	Identifies a variable in text; replace this place holder with a specific value or string.
angle brackets	<filename>	Identifies a variable in code; replace this place holder with a specific value or string.
courier	owsctl start wrb	Text to be entered exactly as it appears. Also used for functions.
square brackets	[-c string] [on off]	Identifies an optional item. Identifies a choice of optional items, each separated by a vertical bar (), any one option can be specified.
braces	{yes no}	Identifies a choice of mandatory items, each separated by a vertical bar ().
ellipses	n,...	Indicates that the preceding item can be repeated any number of times.

The term “Oracle Server” refers to the database server product from Oracle Corporation.

The term “**oracle**” refers to an executable or account by that name.

The term “*oracle*” refers to the owner of the Oracle software.

Technical Support Information

Oracle Global Support can be reached at the following numbers:

- In the USA: **Telephone: 1.650.506.1500**
- In Europe: **Telephone: +44 1344 860160**
- In Asia-Pacific: **Telephone: +61. 3 9246 0400**

Please prepare the following information before you call, using this page as a check-list:

- ☐ your CSI number (if applicable) or full contact details, including any special project information
- ☐ the complete release numbers of the Oracle Application Server and associated products
- ☐ the operating system name and version number
- ☐ details of error codes and numbers and descriptions. Please write these down as they occur. They are critical in helping WWCS to quickly resolve your problem.
- ☐ a full description of the issue, including:
 - **What** - What happened? For example, the command used and its result.
 - **When** -When did it happen? For example, during peak system load, or after a certain command, or after an operating system upgrade.
 - **Where** -Where did it happen? For example, on a particular system or within a certain procedure or table.
 - **Extent** - What is the extent of the problem? For example, production system unavailable, or moderate impact but increasing with time, or minimal impact and stable.
- ☐ Keep copies of any trace files, core dumps, and redo log files recorded at or near the time of the incident. WWCS may need these to further investigate your problem. For a list of trace and log files, see “Configuration and Log Files” in the *Administration Guide*.

For installation-related problems, please have the following additional information available:

- ☐ listings of the contents of \$ORACLE_HOME (Unix) or %ORACLE_HOME% (NT) and any staging area, if used.

-
- ❑ installation logs (**install.log**, **sql.log**, **make.log**, and **os.log**) typically stored in the **\$ORACLE_HOME/orainst** (Unix) or **%ORACLE_HOME%\orainst** (NT) directory.

Documentation Sales and Client Relations

In the United States:

- To order hardcopy documentation, call Documentation Sales: **1.800.252.0303**.
- For shipping inquiries, product exchanges, or returns, call Client Relations: **1.650.506.1500**.

In the United Kingdom:

- To order hardcopy documentation, call Oracle Direct Response: **+44 990 332200**.
- For shipping inquiries and upgrade requests, call Customer Relations: **+44 990 622300**.

Reader's Comment Form

Oracle Application Server Developer's Guide: PL/SQL and ODBC Applications

Part No. A66958-02

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this publication. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most about this manual?

If you find any errors or have suggestions for improvement, please indicate the topic, chapter, and page number below:

Please send your comments to:

Oracle Application Server Documentation Manager
Oracle Corporation
500 Oracle Parkway
Redwood Shores, CA 94065

If you would like a reply, please provide your name, address, and telephone number below:

Thank you for helping us improve our documentation.

Part I

PL/SQL Cartridge

PL/SQL Cartridge Overview

The PL/SQL cartridge provides an environment that enables users to use their browsers to invoke PL/SQL procedures stored in Oracle databases. The stored procedures can retrieve data from tables in the database, and generate HTML pages that include the data to return to the client browser.

PL/SQL is Oracle's procedural language extension to SQL, the standard data access language for relational databases.

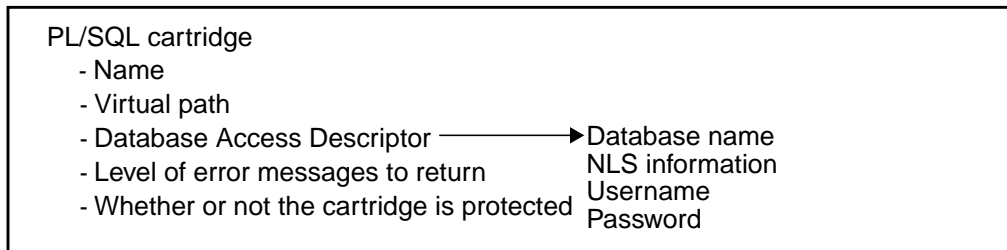
Contents

- [Configuration Information](#)
- [Communication Path](#)
- [POST and GET Methods](#)

Configuration Information

The PL/SQL cartridge enables you to build applications that allow users to run stored procedures in Oracle databases. You provide the cartridge with configuration information as shown in Figure 1-1.

Figure 1–1 PL/SQL cartridge configuration information



Each PL/SQL cartridge is associated with a database access descriptor (DAD), which is a named set of configuration values used for database access. A DAD specifies information, such as:

- The database name or the SQL*Net V2 service name
- The ORACLE_HOME directory
- NLS configuration information, such as language, sort type, and date language

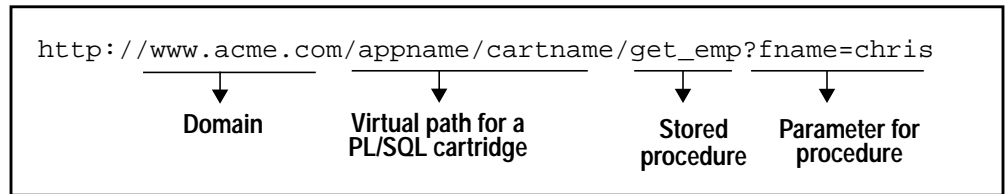
You can also specify username and password information in a DAD; if they are not specified, the user will be prompted to enter a username and password when the URL is invoked.

The database connection information is placed in DADs so that multiple cartridges can use the same DAD. This enables you to define a DAD for each database to which you want to connect, since it is the DAD that specifies the database. You need different PL/SQL cartridges if you want to return different levels of error information or different transaction parameters.

Note to 3.0 Users: In Oracle Application Server 4.0, the PL/SQL agent concept is no longer used. The configuration information in PL/SQL agents has been moved to the cartridge itself.

Communication Path

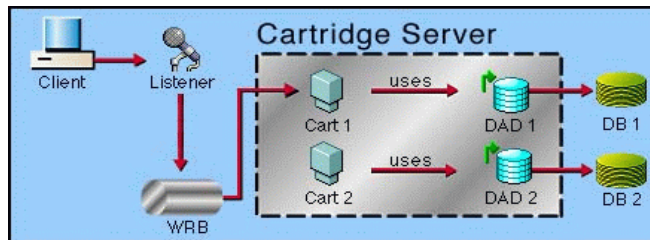
A PL/SQL cartridge is also associated with a virtual path. You use this virtual path to invoke the PL/SQL cartridge. The URL can also contain values for any parameters required by the stored procedure. Figure 1–2 shows the parts of a URL:

Figure 1–2 Breakdown of a URL for a PL/SQL cartridge

You group one or more PL/SQL cartridges, each with its own configuration values, into an “application.” Applications enable you to manage the set of cartridges as a whole, because the cartridges will be running within the same cartridge server process. For example, you can stop the cartridges by stopping the cartridge servers, configure the logging level for all cartridges in an application, and specify the number of instances of each cartridge that can run in a process.

The following sequence of steps takes place when Oracle Application Server receives a request for a PL/SQL cartridge (Figure 1–3):

1. The listener receives a request from a client, and determines who should handle it. In this case, it forwards the request to the web request broker (WRB) because the request is for a cartridge.
2. From the virtual path, the WRB determines the PL/SQL cartridge that should handle the request, and sends the request to a cartridge server that is running the application.
3. In the cartridge server, the PL/SQL cartridge uses the DAD’s configuration values to determine which database to connect and how to set up the PL/SQL client configuration.
4. The PL/SQL cartridge connects to the database, prepares the call parameters, and invokes the procedure in the database.
5. The procedure generates an HTML page, which can include dynamic data accessed from tables in the database as well as static data.
6. The output from the procedure is returned via the response buffer back to the PL/SQL cartridge and the client.

Figure 1–3 Connecting to an Oracle database from a PL/SQL cartridge

The stored procedure that the cartridge invokes should return HTML data back to the client. To simplify this task, the PL/SQL cartridge comes with the PL/SQL Web Toolkit, which is a set of packages that you can use in your stored procedure to get information about the request, construct HTML tags, and return header information to the client. You install the toolkit in a common schema so that all users can access it.

To configure PL/SQL cartridges and DADs, use the Oracle Application Server Manager, which is a set of HTML forms. Use these forms to enter configuration information, such as virtual paths for the PL/SQL cartridge, the SQL*Net V2 service name for the DAD, and the error reporting.

When designing your applications, you must consider security issues. You have to design your application such that unauthorized users do not have access to the application, and authorized users can run the application only in the proper context. See Chapter 5, “Authentication and Security” for details.

POST and GET Methods

POST and GET methods in the HTTP protocol instruct browsers how to pass parameter data (usually in the form of name-value pairs) to applications. The parameter data are usually generated by HTML forms.

Oracle Application Server applications can use either method. The method that you use is as secure as the underlying transport protocol (http or https).

When you use the POST method, parameters are passed in the request body, and when you use the GET method, parameters are passed using the query string. These methods are described in the HTTP 1.1 specification, which is available at the W3C web site, <http://www.w3.org/Protocols/HTTP/1.1/draft-ietf-http-v11-spec-rev-01.txt>.

The limitation of the GET method is that the length of the value in a name-value pair cannot exceed the maximum length for the value of an environment variable,

as imposed by the underlying operating system. In addition, operating systems have a limit on how many environment variables you can define.

Generally, if you are passing large amounts of parameter data to the server, you should use the POST method instead.

This section provides a step-by-step guide on creating and invoking a simple application that displays the contents of a database table as an HTML table. The application consists of one PL/SQL cartridge. The cartridge invokes a stored procedure that calls functions and procedures defined in the PL/SQL Web Toolkit.

This tutorial steps you through the following tasks:

1. [Creating and Loading the Stored Procedure onto the Database](#)
2. [Creating an Application and its Components](#)
3. [Reloading](#)
4. [Creating an HTML Page to Invoke the Application](#)

This tutorial assumes the following:

- You can log in as the “admin” user for Oracle Application Server. This is required because you will be adding new settings to the server configuration.
- The database to which you will be connecting already has the PL/SQL Web Toolkit installed. See [“PL/SQL Web Toolkit Installation” on page 4-2](#).
- You have the “scott” schema in your database. The PL/SQL cartridge logs into the database using scott/tiger as the username and password. If you do not have the “scott” schema, you can use an existing schema on your database, or you can create the “scott” schema using the “CREATE SCHEMA” command.

A schema can be thought of as a user account: it is a collection of database objects such as tables, views, procedures, and functions, and each object in the schema can access other objects in the same schema.

1. Creating and Loading the Stored Procedure onto the Database

The stored procedure that the application invokes is `current_users` (defined below). The procedure retrieves the contents of the `all_users` table and formats it as an HTML table.

To create the stored procedure, save the text of the procedure in a file called **current_users.sql**, and then run Oracle Server Manager to read and execute the statements in the file.

1. Type the following lines and save it in a file called **current_users.sql**. The `current_users` procedure retrieves the contents of the `all_users` table and formats it as an HTML table.

```
create or replace procedure current_users
AS
    ignore boolean;
BEGIN
    http.htmlopen;
    http.headopen;
    http.title('Current Users');
    http.headclose;
    http.bodyopen;
    http.header(1, 'Current Users');
    ignore := owa_util.tablePrint('all_users');
    http.bodyclose;
    http.htmlclose;
END;
/
show errors
```

This procedure uses functions and procedures from the `http` and `owa_util` packages to generate the HTML page. For example, the `http.htmlopen` procedure generates the string `<html>`, and `http.title('Current Users')` generates `<title>Current Users</title>`.

The [owa_util.tablePrint function](#) queries the specified database table, and formats the contents as an HTML table.

2. Start up Server Manager in line mode. `ORACLE_HOME` is the directory that contains the Oracle database files.

```
prompt> $ORACLE_HOME/bin/svrmgr1
```

3. Connect to the database as “scott”. The password is “tiger”.

```
SVRMGR> connect scott/tiger
```

4. Load the `current_users` stored procedure from the **current_users.sql** file. You need to provide the full path to the file if you started up Server Manager from a directory different than the one containing the **current_users.sql** file.





```
SVRMGR> @  
Name of script file: current_users.sql
```

5. Exit Server Manager.

```
SVRMGR> exit
```

2. Creating an Application and its Components

When you create an application, you also create its components: PL/SQL cartridges and DADs (database access descriptor). You create these components using the Oracle Application Server Manager administration forms.

1. Start up your browser and display the top-level administration page for Oracle Application Server. You should see “OAS Sites” at the top of the left frame.
2. Click  to display the sites.
3. Click the  next to a site name to display the components on the site. You should see “HTTP Listeners”, “Oracle Application Server”, and “Applications”.
4. Click “Applications” to display the applications in the right frame. Do not click the  next to Applications because you will see a list of applications for the site in the left frame, instead of Applications in the right frame.
5. On the applications page in the right frame, click . The Add Application dialog opens.
6. In the Add Application dialog:
 - Application Type: select PLSQL.
 - Configure Mode: select Manually, which enables you to enter configuration data using dialog boxes. The other option, From File, assumes that you have already entered the configuration data for the application in a file.
 - Click Apply.

The Add Application dialog box appears on the screen.

7. In the Add Application dialog:
 - Application Name: enter “simpleApp1”. This name is used to identify your application.

- Display Name: enter “simpleApp1”. This name is used in the administration forms.
 - Application Version: enter “1.0”.
 - Click Apply.

When you click Apply, you get a Success dialog box, which contains a button that enables you to add cartridges to the application.
8. In the Success dialog box, click the Add Cartridges to Application button. This displays the Add PLSQL Cartridge dialog.
9. In the Add PLSQL Cartridge dialog:
- Cartridge Name: enter “cart1”. This name is used to identify your PL/SQL cartridge in your “simpleApp1” application.
 - Display Name: enter “cart1”. This name is used in the administration forms.
 - Virtual Path: enter **/simpleApp1/cart1** as the virtual path for your PL/SQL cartridge.
 - Physical Path: this field shows **%ORAWEB_HOME%/bin** as the physical path. Leave this field as it is.
 - Click the Create New DAD button. The Database Access Descriptor dialog box appears on the screen. A DAD specifies connection information such as the database to which you want to connect, and the username and password to use to log into the database. Users will use this same DAD later to run the stored procedure.
10. In the Database Access Descriptor dialog:
- DAD Name: enter “scott”.
 - Database User: enter “scott”.
 - Database User Password and Confirm Password: enter “tiger”.
 - Database Location (Host): enter the machine running the database.
 - Database Name (ORACLE_SID) or Database Network Service Name: if the database is running on the same machine as the application, specify the database’s ORACLE_SID. If you are accessing a database running on another machine, enter the SQL*Net v2 connect string in the Database Network Service Name instead.
 - Create Database User: leave unselected.

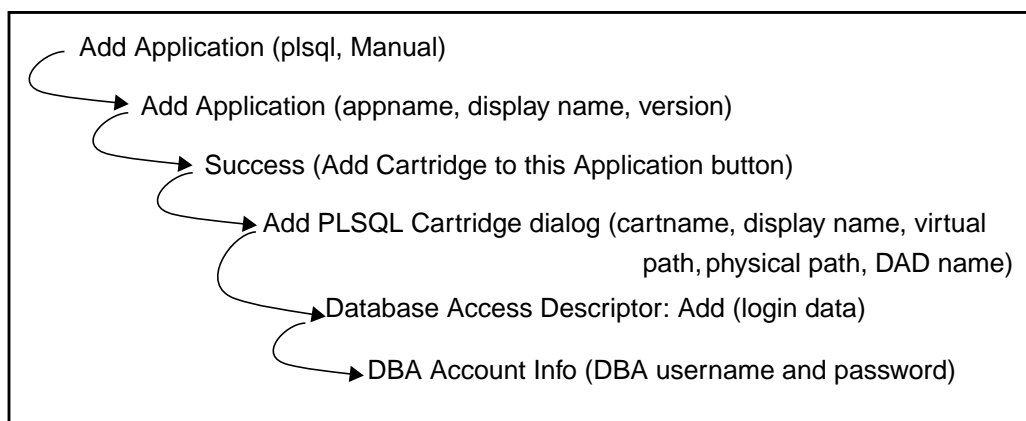
- Change Database Password: leave unselected.
- Store the username and password in the DAD: select this option.
- Click Apply.

11. In the DBA Account Info dialog:

- DBA Username: enter the DBA username
- Password and Confirm Password: enter the DBA password
- Click Apply.


The following figure summarizes the dialog boxes that you completed. The fields in the dialog boxes are listed in parentheses.

Figure 2–1 Dialogs to create a PL/SQL application, PL/SQL cartridge, and DAD





3. Reloading

After reconfiguring Oracle Application Server, you have to reload the server for the new configuration to take effect. To reload the application server:

1. Select Applications in the Oracle Application Server Manager. This displays a list of applications in the right frame.
2. Select ALL.
3. Click  to reload the application server.

You also have to stop and restart the listener for the new virtual path to take effect. To stop and restart the listeners:

1. Click Http Listeners in the navigational tree to display the list of listeners.
2. Select the listeners that you want to stop. You must select all the listeners that will be handling the request for the application.
3. Click  to stop the listeners.
4. Click  to restart the listeners.

Note: Whenever you stop and restart the Oracle Application Server components, you also have to stop and restart the listeners.

4. Creating an HTML Page to Invoke the Application

To run the `current_users` procedure, enter the following URL in your browser:

`http://<host>:<port>/simpleApp1/cart1/current_users`

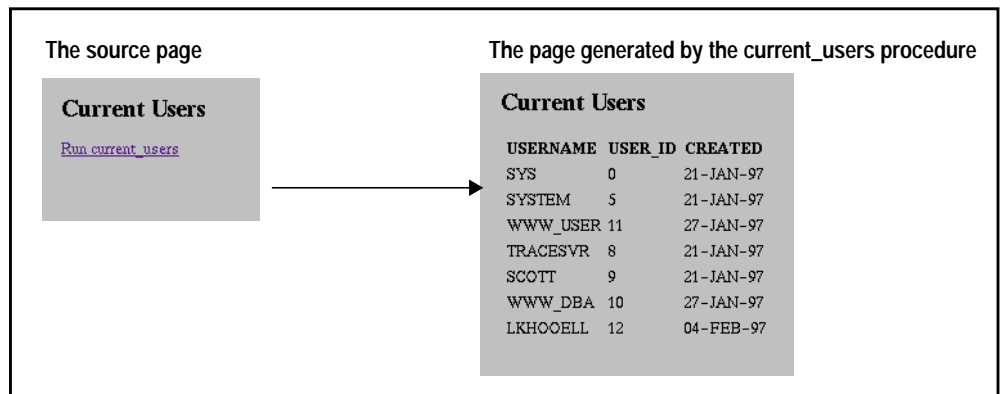
host and *port* identify the listener that knows about the cartridge. This is any listener on the application server except the node manager listener (which runs on port 8888 by default). For example, you can use the administration utility listener, which runs on port 8889 by default.

It is more common, however, to invoke the procedure from an HTML page. For example, the following HTML page has a link that calls the URL.

```
<HTML>
<HEAD>
<title>Current Users</title>
</HEAD>

<BODY>
<H1>Current Users</H1>
<p><a href="http://hal.us.oracle.com:9999/simpleApp1/cart1/current_users">Run
current_users</a>
</BODY>
</HTML>
```

Figure 2–2 shows the source page (the page containing the link that invokes the stored procedure), and the page that is generated by the `current_users` stored procedure.

Figure 2–2 The source page and the dynamically generated page in the tutorial

Adding and Invoking PL/SQL Applications

This chapter describes how to add PL/SQL applications to Oracle Application Server and how to invoke them from browsers.

Contents

- [Adding PL/SQL Applications](#)
- [Configuring PL/SQL Applications](#)
- [Invoking a PL/SQL Cartridge](#)
- [Overloaded Procedures](#)
- [Variables with Multiple Values](#)
- [Flexible Parameter Passing](#)
- [Positional Parameters](#)
- [Executing SQL Files](#)
- [Life Cycle of the PL/SQL Cartridge](#)




Adding PL/SQL Applications

To add PL/SQL applications to Oracle Application Server, you perform these steps:

1. Add the application.
2. Add cartridge(s) to the application
3. Add database access descriptors (DADs) to the application server. DADs can be shared by more than one cartridge. You can add DADs when you add cartridges, or you can add them beforehand.

Adding Applications and Cartridges

You can add applications and cartridges to Oracle Application Server by following these steps:

1. Start up your browser and display the top-level administration page for Oracle Application Server.
2. Click the  next to a site name to display the components on the site. You should see “Oracle Application Server”, “HTTP Listeners”, and “Applications”.
3. Click “Applications” to display the applications in the right frame. Do not click the  next to Applications because you will see a list of applications for the site in the left frame, instead of Applications in the right frame.
4. On the applications page in the right frame, click . This pops up the Add Application dialog.
5. In the Add Application dialog:
 - Application Type: select PL/SQL.
 - Configure Mode: select Manually, which enables you to enter configuration data using dialog boxes. The other option, From File, assumes that you have already entered the configuration data for the application in a file.
 - Click Submit.

This displays the Add Application dialog.

6. In the Add Application dialog:
 - Application Name: enter the name that the server uses to identify your application.
 - Display Name: enter the name that is used in the administration forms.
 - Application Version: enter a version number for your application.
 - Click Apply.

When you click Apply, you get a Success dialog box, which contains a button that enables you to add PL/SQL cartridges to the application.

7. In the Success dialog box, click the Add Cartridges to Application button. This displays the Add PLSQL Cartridge dialog.
8. In the Add PLSQL Cartridge dialog:
 - Cartridge Name: enter the name that the server uses to identify your PL/SQL cartridge in your application.


- **Display Name:** enter the name that is used in the administration forms.
- **Virtual Path:** enter a virtual path for your PL/SQL cartridge. The default virtual path for the PL/SQL cartridge is `/<appName>/<cartName>`.
Users specify this virtual path in URLs to invoke the PL/SQL cartridge. The virtual path maps onto the location of your class files.
- **Physical Path:** leave this field as it is if your PL/SQL cartridge is running stored procedures. If your cartridge is running PL/SQL source files (".sql" extension), enter the full path of the directory that the PL/SQL source files. See [“Executing SQL Files” on page 3-18](#) for details on executing PL/SQL source files.
- **DAD Name:** select the name of the database access descriptor that is used by this PL/SQL cartridge. If the DAD does not exist, you can create it by clicking the Create New DAD button.
- **Create New DAD button.** This displays the Add DAD dialog. A DAD specifies database connection information such as the database to which you want to connect, and the username and password to use to log into the database. When users invoke the PL/SQL cartridge, the cartridge uses the information in the DAD to connect to the database run the stored procedure. See the next section for more details on DADs.

Creating Database Access Descriptors (DADs)

DADs provide the information on how to connect to the Oracle database containing the stored procedures that you want to execute. Each PL/SQL cartridge is associated with one DAD.

DADs contain information such as the name of the database (the `ORACLE_SID` or the network service name) and the host machine of the database. You can also provide database username and password information, but if you do not, the user will be prompted for this information.

To add a DAD to the application server, you use the Add DAD dialog. You can access the dialog from two places:

- From the Create New DAD button in the Add PLSQL Cartridge dialog.
- From the  icon in the DB Access Descriptor form, which is located under Oracle Application Server in the navigational tree.

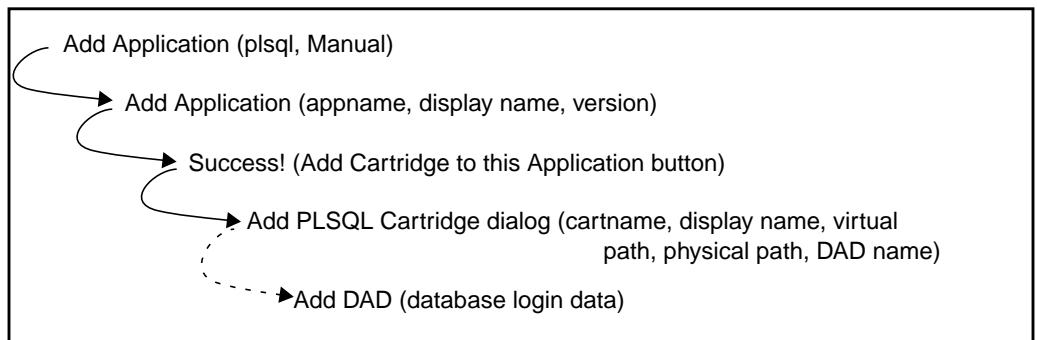
In the Add DAD dialog:

- **DAD Name:** enter the name of the DAD.

- Database User: enter the username that will be used to log into the database.
- Database User Password and Confirm Password: enter the password for the username.
- Database Location (Host): enter the machine that contains the files for the database's ORACLE_HOME.
- Database Name (ORACLE_SID): if your database is running on the same machine as the application, specify the database's ORACLE_SID.
- Connect String: if you are accessing a database running on another machine, enter the SQL*Net v2 connect string.
- Create Database User: select if the specified username does not exist in the database. If you select this option and you have installed the PL/SQL Web Toolkit in this release (not from a previous release), the new user is given proper privileges to access the Content service database objects.
- Change Database Password: select if you want to change the username's password.
- Store the username and password in the DAD: if you save the username and password with the DAD, all cartridges that use this DAD will log into the database using these values, and the user will not be prompted. If you do not save the username and password information with the DAD, the user will be prompted for the username and password information when the cartridge needs to log into the database.
- Click Apply.

Note: To get your application to appear in the navigational tree, shift-click the browser's Reload button.

The following figure summarizes the dialog boxes that you completed. The fields in the dialog boxes are listed in parentheses.

Figure 3–1 Dialogs to create a PL/SQL application, PL/SQL cartridge, and DAD

Starting and Stopping the Application Server

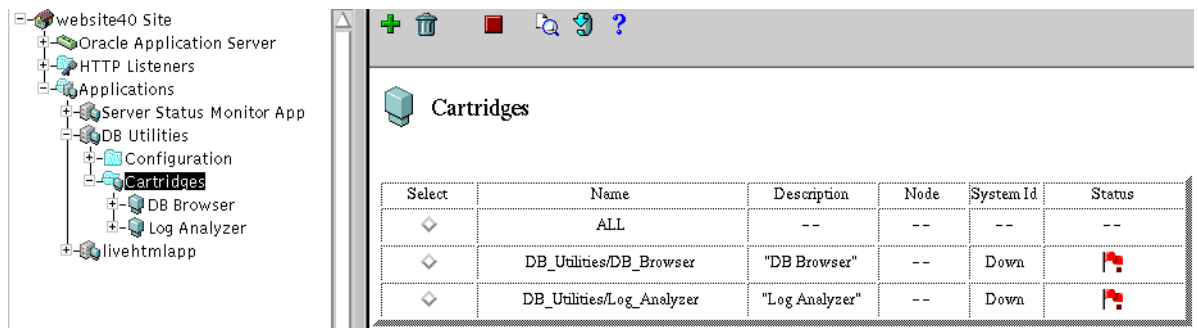
After adding applications to the application server, you have to stop and restart the listeners and the web request broker (WRB). See “3. Reloading” in Chapter 2, “Tutorial” for details.


Adding Cartridges to an Existing Application

A PL/SQL application can have one or more cartridges. You need more than one cartridge in a PL/SQL application if the application needs to access more than one database.

To add a cartridge to a PL/SQL application:

1. Select “Cartridges” under the PL/SQL application to which you want to add cartridges in the navigational tree.

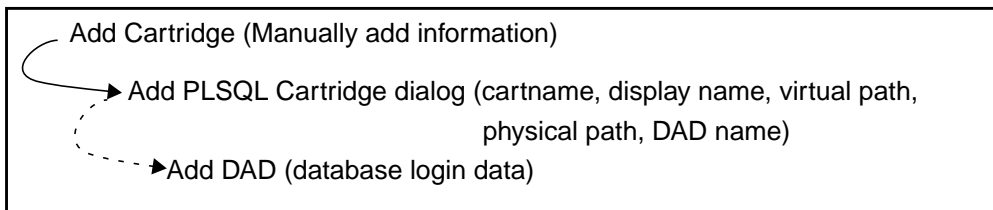
Figure 3–2 Adding PL/SQL cartridges to an existing application

2. Click  to display the Add Cartridge dialog.
3. In the Add Cartridge dialog:
 - Configure Mode: select Manually.
 - Click Apply, which displays the Add PLSQL Cartridge dialog.
4. In the Add PLSQL Cartridge dialog:
 - Cartridge Name: enter the name that the server uses to identify your PL/SQL cartridge in your application.
 - Display Name: enter the name that is used in the administration forms.
 - Virtual Path: enter a virtual path for your PL/SQL cartridge. The default virtual path for the PL/SQL cartridge is /<appName>/<cartName>.
 - Physical Path: leave this field as it is if your PL/SQL cartridge is running stored procedures. If your cartridge is running PL/SQL source files (".sql" extension), enter the full path of the directory that the PL/SQL source files. See ["Executing SQL Files" on page 3-18](#) for details on executing PL/SQL source files.
 - DAD Name: select the database access descriptor that is used by this PL/SQL cartridge. DADs specify the database to which the cartridge connects. If the DAD does not exist, you can create it by clicking the Create New DAD button.
 - Click Apply.

Note: To get your new cartridge to appear in the navigational tree, shift-click the browser's Reload button.

The following figure summarizes the dialog boxes that you completed. The fields in the dialog boxes are listed in parentheses.

Figure 3–3 *Dialogs to add PL/SQL cartridges*



Configuring PL/SQL Applications

The configuration forms are divided into two sections: application configuration and cartridge configuration. Forms in the application configuration section contain parameters that apply for the entire application, while forms in the cartridge configuration section contain parameters that apply to a particular cartridge.

Application Configuration

Application configuration parameters are described in Chapter 6, “Application Administration” in the *Oracle Application Server Administration Guide* because they are the same for all types of applications.

Cartridge Configuration

For PL/SQL cartridges, the cartridge configuration section contains two forms: the Virtual Paths form and the PL/SQL Parameters form.

Virtual Paths Form

Each PL/SQL cartridge is mapped to a virtual path, and you use this virtual path in URLs to invoke the cartridge. The last component of virtual paths for PL/SQL cartridges specifies the stored procedure to run. For example, if you map the virtual path `/hr/benefits` to a PL/SQL cartridge, a URL of `/hr/benefits/intro` would execute the “intro” stored procedure. See [“Invoking a PL/SQL Cartridge” on page 3-8](#) for URL details.

The default virtual path for a PL/SQL cartridge is `/<appName>/<cartName>`. You can change this virtual path using the Virtual Paths form.

In version 3.0 of Oracle Web Application Server, the virtual path had to contain the name of the PL/SQL Agent. In version 4.0, the virtual path can be anything you like. Each PL/SQL cartridge is associated with a DAD, which contains database connection information.

To make virtual paths easier to maintain, you can adopt your own convention; for example, you can use the default convention of `/appName/cartname`, where *appName* specifies the name of the PL/SQL application, and *cartname* specifies the name of the PL/SQL cartridge. Note that there is no requirement to have two components in the virtual path. You can have as many components as you wish.

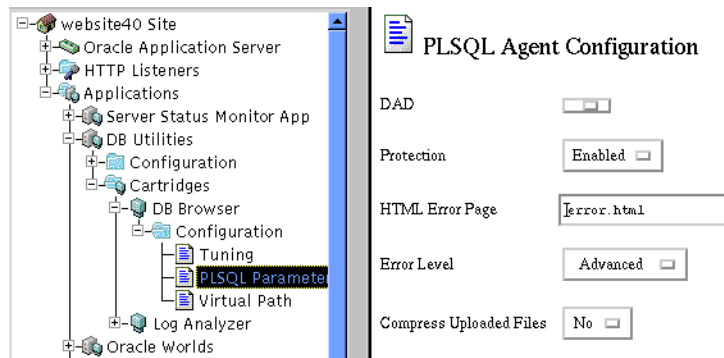
For more information on the Virtual Paths form, see Chapter 6, “Application Administration” in the *Oracle Application Server Administration Guide*.

PL/SQL Parameters Form

The PL/SQL Parameters form enables you to specify the following:

- The DAD associated with the PL/SQL cartridge
- Whether or not protection is enabled
- The HTML page to display if an error occurs
- How much error information to display to the user if an error occurs
- Whether or not uploaded files are stored in compressed format

Figure 3–4 PL/SQL Parameters form



Invoking a PL/SQL Cartridge

You invoke a PL/SQL cartridge by giving a URL in the browser or in an HTML page.

URL Format

To invoke a PL/SQL cartridge, the URL must be in the following format:

`http://hostname[:port]/virtual_path/[package.]proc_name[?QUERY_STRING]`

where:

- *hostname* specifies the machine where the application server is running.
- *port* specifies the port at which the application server is listening. If omitted, port 80 is assumed.

- *virtual_path* specifies a virtual path mapped to the PL/SQL cartridge. The virtual path can contain any number of components, and the string “*plsql*” does not have to appear anywhere in it. (This was not a requirement in 3.0, either.)
- *package* specifies the package (if any) that contains the procedure. If omitted, the procedure is stand-alone.
- *proc_name* specifies the stored procedure to run. This must be a procedure and not a function.
- *QUERY_STRING* specifies parameters (if any) for the stored procedure. The string follows the format of the GET method. For example, multiple parameters are separated with the & character, and space characters in the values to be passed in are replaced with the + character. If you use HTML forms to generate the string (as opposed to generating the string yourself), the formatting will be done automatically for you.

For example, if a browser sends the following URL:

```
http://www.acme.com:9000/mycartx/  
get_emp?fname='john'&lname='doe'&role='office+manager'
```

the application server running on **www.acme.com** and listening at port 9000 would handle the request. When the listener receives the request, it passes the request to the WRB because it sees that the **/mycartx** virtual path is configured to call a PL/SQL cartridge in a PL/SQL application. The WRB sends the request to a cartridge server that is running the PL/SQL application.

The PL/SQL cartridge instance connects to the database using the DAD associated with the cartridge and runs the `get_emp` stored procedure. The *fname* parameter of the procedure gets the value `john`, the *lname* parameter gets the value `doe`, and the *role* parameter gets the value “`office manager`”. The space character is put back in before the stored procedure sees the value.

Generally, you need not be concerned with the order in which PL/SQL parameters are given in the URL or the HTTP header, because the parameters are passed by name. However, there are two exceptions to this rule:

- You could have multiple parameters of the same name. This might happen if you have an HTML form that contains multiple elements with the same name. In this case, the parameter is passed to the PL/SQL procedure as a PL/SQL table. See [“Variables with Multiple Values” on page 3-12](#).
- You can pass parameter by position to the PL/SQL cartridge. See [“Positional Parameters” on page 3-17](#).

Caching Procedure Descriptions

Before executing a procedure, the PL/SQL cartridge gets the description of the requested procedure from the database. This is done to check the type and number of arguments of the procedure to be invoked. However, the procedure descriptions are not expected to change in a production environment, and hence, one cartridge-to-database network round trip can be avoided by caching the procedure description.

The PL/SQL cartridge caches the procedure descriptions by default. Any modifications to the procedure prototype while its description is cached by a cartridge server process will put the cache out of synch. You need to restart the PL/SQL cartridge server process to view the modified procedure description.

If for some reason, the procedure prototype needs frequent changes, restarting the PL/SQL cartridge server process every time is not a viable option. In such cases, you should turn off caching of the procedure descriptions. You can turn off caching of procedure descriptions by adding the following line manually under the `RUNTIME . PLSQL` section in the **wrb.app** file:

```
Cache_Proc_Desc      =      false
```

Note that updates to **wrb.app** should be done only after Oracle Application Server is shut down.

Overloaded Procedures

PL/SQL supports overloading, where multiple subprograms (procedures or functions) have the same name, but differ in the number, the order, or the data type family of the parameters. When you call an overloaded subprogram, the PL/SQL compiler determines which subprogram to call based on the data types passed. PL/SQL allows you to overload local or packaged subprograms; standalone subprograms cannot be overloaded. See the Oracle8 documentation for more information on PL/SQL overloading.

In addition to PL/SQL's restrictions on overloading, the PL/SQL cartridge places one more restriction: you must give the parameters different names for overloaded subprograms that have the same number of parameters. The reason for this is that HTML data is not associated with data types, and this makes it impossible for the cartridge to know which version of the subprogram to call. For example, PL/SQL allows you to define the following two procedures, but you will get an error when you try to use them with the PL/SQL cartridge because the parameter names are the same:

```
-- legal PL/SQL, but not for the PL/SQL cartridge
CREATE PACKAGE my_pkg AS
    PROCEDURE my_proc (val IN VARCHAR2);
    PROCEDURE my_proc (val IN NUMBER);
END my_pkg;
```

To avoid the error, name the parameters differently. For example:

```
-- legal PL/SQL and also works for the PL/SQL cartridge
CREATE PACKAGE my_pkg AS
    PROCEDURE my_proc (valvc2 IN VARCHAR2);
    PROCEDURE my_proc (valnum IN NUMBER);
END my_pkg;
```

To invoke the first version of the procedure, the URL looks something like:

```
http://www.acme.com/mycart/my_pkg.my_proc?valvc2=input
```

To invoke the second version of the procedure, the URL looks something like:

```
http://www.acme.com/mycart/my_pkg.my_proc?valnum=34
```

Overloading and PL/SQL Arrays

If you have overloaded PL/SQL procedures where the parameter names are identical, but where the data type is `owa_util.ident_arr` for one procedure and a scalar type for another procedure, the PL/SQL cartridge can still distinguish between the two procedures. For example, if you have the following procedures:

```
CREATE PACKAGE my_pkg AS
    PROCEDURE my_proc (val IN VARCHAR2); -- scalar data type
    PROCEDURE my_proc (val IN owa_util.ident_arr); -- array data type
END my_pkg;
```

Each of these procedures has a single parameter of the same name, “val”.

When the PL/SQL cartridge gets a request that has only one value for the *val* parameter, it invokes the procedure with the scalar data type. When it gets a request with more than one value for the *val* parameter, it then invokes the procedure with the array data type.

Example 1: if you send the following URL:

```
.../my_proc?val="john"
```

the scalar version of the procedure would execute.

Example 2: if you send the following URL:

```
.../my_proc?val="john"&val="sally"
```

the array version of the procedure would execute.

If you want to ensure that the array version of the procedure runs, you can use hidden form elements on your HTML page to send dummy values, which are checked and discarded in your procedure. See the following section for an example.

Variables with Multiple Values

A browser can return, to the server, variables that contain multiple values. For example, an HTML form that uses the `SELECT` element allows users to select one or more values from a given set. You can also have different form elements with the same value for the `NAME` attribute, in which case the values for these elements will be returned on the same variable name.

The PL/SQL cartridge handles multi-value variables by storing the values in a PL/SQL table. This enables you to be flexible about how many values the user can pick, and it makes it easy for you to process the user's selections as a unit. Each value is stored in a row in the PL/SQL table, starting at index 1. The first value (in the order that it appears in the query string) of a variable that has multiple values is placed at index 1, the second value of the same variable is placed at index 2, and so on. If the order of the values in the PL/SQL table is significant in your procedure, you need to determine the order in which the variables appear in the query string.

If you do not have variables with multiple values, you do not have to worry about the order in which the variables appear, because their values are passed to the procedure's parameters by name, not by position.

The PL/SQL tables used as parameters in the PL/SQL cartridge environment must have a base type of `VARCHAR2`. Oracle can convert `VARCHAR2` to other data types such as `NUMBER`, `DATE`, or `LONG`. The maximum length of a `VARCHAR2` variable is 32K.

If you cannot guarantee that at least one value will be submitted for the PL/SQL table (for example, the user can select zero options), use a hidden form element to provide the first value. Not providing a value for the PL/SQL table produces an error, and you cannot provide a default value for a PL/SQL table.

The following example passes a multi-value parameter into a PL/SQL table. The form contains a `SELECT` element and a set of checkbox elements. Note the two hidden elements: one is used for the `SELECT` element, and the other for the checkboxes. This is the HTML that creates the form:


```
<html>
<head>
<title>Multivalue Example</title>
</head>

<body>
<h1>Multivalue Example</h1>

<p>This form shows how variables with multiple values are
handled by the PL/SQL Cartridge. The form has one SELECT
element and a set of checkbox elements.

<form method="PUT" action="/mycart/dept_machine">

<input type="hidden" name="departments" value="no_value">
<input type="hidden" name="machines" value="no_value">

<p>Select the departments in which you want to search:
<p>
<select name="departments" multiple>
  <option>Benefits
  <option>Marketing
  <option>Finance
  <option>Sales
  <option>Engineering
  <option>QA
  <option>Customer Support
</select>

<p>Select the machine type:<br>
<input type="checkbox" name="machines" value="PC">PC<br>
<input type="checkbox" name="machines" value="Mac">Mac<br>
<input type="checkbox" name="machines" value="Sun">Sun<br>
<input type="checkbox" name="machines" value="Other">Other<br>

<p><input type="submit" value="Search">
</form>
</body>
</html>
```

Figure 3–5 shows the form as it appears in a browser:

Figure 3–5 Form passing in multiple values

Multivalue Example

This form shows how variables with multiple values are handled by the PL/SQL Cartridge. The form has one SELECT element and a set of checkbox elements.

Select the departments in which you want to search:

- ☐ Benefits
- ☐ Marketing
- ☐ Finance
- ☐ Sales
- ☐ Engineering
- ☐ QA
- ☐ Customer Support

Select the machine type:

- ☐ PC
- ☐ Mac
- ☐ Sun
- ☐ Other

When the user clicks Search, the `dept_machine` procedure runs on the database. The procedure simply returns an HTML page that lists the user's selections. Note that the loop counter starts at index 2 because the hidden element values are in index 1 in the PL/SQL tables. When the procedure prints out the number of rows in the PL/SQL tables, it subtracts one to avoid counting the hidden row.

```
create or replace procedure dept_machine (
    departments IN owa_util.ident_arr,
    machines    IN owa_util.ident_arr )
IS
    counter INTEGER;
    ct      INTEGER;
BEGIN
    http.htmlopen;
    http.headopen;
    http.title('Dept and Machines Results');
    http.headclose;

    http.bodyopen;
    http.header(1, 'Dept and Machines Results');

    ct := departments.COUNT - 1;
```

```

http.print('The "departments" PL/SQL table has ' || ct || ' rows.');
```

```

http.print('You selected:');
http.ulistOpen;
FOR counter IN 2 .. departments.COUNT LOOP
    http.listItem(departments(counter));
END LOOP;
http.ulistClose;

ct := machines.COUNT - 1;
http.print('The "machines" PL/SQL table has ' || ct || ' rows.');
```

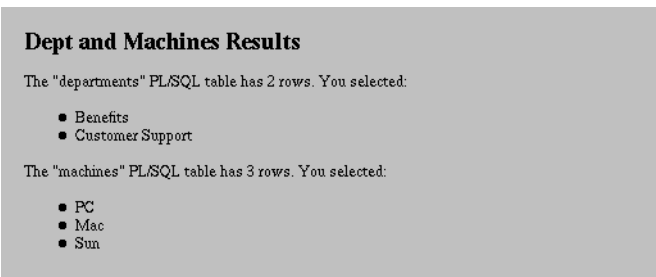
```

http.print('You selected:');
http.ulistOpen;
FOR counter IN 2 .. machines.COUNT LOOP
    http.listItem(machines(counter));
END LOOP;
http.ulistClose;
http.paragraph;

http.bodyclose;
http.htmlclose;
END;
/
show errors
```

For example, if the user selects “Benefits” and “Customer Support” from the SELECT element, and “PC”, “Mac”, and “Sun” from the checkbox, the procedure returns an HTML page that looks like the page in Figure 3–6.

Figure 3–6 Generated page from the multi-value example



Note: OCI has a limitation of 2000 bytes on the size of VARCHAR2. Please keep this limitation in mind while developing your application.

Flexible Parameter Passing

You can have HTML forms from which users can select any number of elements. If these elements have different names, you would have to create overloaded procedures to handle each possible combination, or, alternatively, you could insert hidden form elements to ensure that the names in the query string are consistent each time, regardless of which elements the user chooses.

Oracle Application Server 4.0 makes this easier by enabling you to define a procedure that the PL/SQL cartridge invokes when you do not have a procedure that matches the names in the query string. The procedure is passed all the name-value pairs in the query string, and it has the following signature:

```
proc_name(  
    num_entries    IN NUMBER,  
    name_array     IN OWA.vc_arr,  
    value_array    IN OWA.vc_arr,  
    reserved       IN OWA.vc_arr)
```

where:

proc_name is the name of the PL/SQL procedure that you are invoking.

num_entries specifies the number of name-value pairs in the query string.

name_array specifies the names from the query string.

value_array specifies the values from the query string.

reserved is not used currently. It is reserved for future use.

If you do not define this procedure and the names in the query string do not match any of the procedures, you will get an error message.

Here is an example of a procedure that prints out the name-value pairs in the query string:

```
CREATE or REPLACE PROCEDURE MY_PROC (  
    num_entries    IN NUMBER,  
    name_array     IN OWA.vc_arr,  
    value_array    IN OWA.vc_arr,  
    reserved       IN OWA.vc_arr)  
IS  
BEGIN  
    http.htmlopen;  
    http.headopen;  
    http.title('Unmatched query string example');
```

```

http.headclose;

http.bodyopen;
http.header(1, 'Unmatched query string example');

http.print('Query string has ' || num_entries ||
           ' name-value pairs.');
```

```

http.dlistOpen;
FOR counter IN 1 .. num_entries
LOOP
    http.dlistTerm(name_array(counter));
    http.dlistDef(value_array(counter));
END LOOP;
http.dlistClose;
http.bodyclose;
http.htmlclose;
END;
/
show errors
```

Positional Parameters

When the PL/SQL cartridge receives a request for a stored procedure, it must make two trips to the database: first, it connects to the database to determine if the specified procedure exists. If the procedure does exist, it then makes a second trip to instruct the database to execute the procedure.

If you are certain that the procedure exists in the database, you can save a trip by telling the PL/SQL cartridge to skip the verification step. To do this, prefix the name of the procedure with a ^ character in the URL. This also causes any parameters for the procedure to be matched by position instead of by name.

For example, if you have a procedure called “my_proc” that takes two parameters, *param1* and *param2*, you can invoke it with any of the following URLs:

```

/* usual style */
http://machine/mycart/my_proc?param1=val1&param2=val2

/* bypass procedure verification, parameters are matched by position */
http://machine/mycart/^my_proc?param1=val1&param2=val2

/* bypass procedure verification, parameter names do not have to match */
http://machine/mycart/^my_proc?x=val1&y=val2
```

```
/* for procedures in packages, the ^ character comes before the package name */  
http://machine/mycart/^my_pkg.my_proc?x=val1&y=val2
```

When you use the usual invocation style (that is, without the ^), parameters are matched by name. When you use the ^ character, parameters are matched by position. This means that you can use any string for the parameter name; you do not have to match it exactly with the name in the database. However, you must match the number of parameters, and you must be careful of the order in which the values appear in the query string.

Note that the flexible parameter passing feature (on page 3-16) does not work with this positional parameter feature. If the number of parameters does not match or if the procedure does not exist, you will get a generic error message that might not pinpoint the error exactly.

Executing SQL Files

In addition to running PL/SQL procedures stored in the database, the PL/SQL cartridge can also run PL/SQL source files from the file system. This feature enables you to execute PL/SQL statements without storing them in the database. You might want to use this feature while prototyping PL/SQL code. This saves you from re-loading procedures into the database each time you edit them.

The file contains an anonymous PL/SQL block, that is, it does not define a function or procedure. The file begins with either a `DECLARE` or a `BEGIN` statement. You need the `DECLARE` statement only if you are using variables in the block. For more details on anonymous blocks, see the PL/SQL documentation.

You need to name your PL/SQL files with a “.sql” extension, because this is how the PL/SQL cartridge distinguishes procedure names from filenames when it reads the URL. Otherwise, the syntax is similar to that of using the cartridge to run stored procedures.

Note that you cannot use `owa_util.showsource` to display the PL/SQL statements in the .sql files because the files are stored in the cartridge rather than in the database.

Note: If you are using the PL/SQL cartridge to run SQL files from the filesystem, the SQL file cannot have a “/” character at the end of the file. If a “/” is present at the end of the file, the PL/SQL cartridge will be unable to execute the specified file.

Location of PL/SQL Source Files

The PL/SQL source files are located in the physical paths that are associated with the virtual paths. For example, if you map the **/test/sql** virtual path to the **%ORAWEB_HOME%/sample/plsql** physical path, the following URL

```
http://machine.domain.name:port/test/sql/mysqlfile.sql
```

would execute PL/SQL statements from the file **%ORAWEB_HOME%/sample/plsql/mysqlfile.sql**. The PL/SQL cartridge configuration information associated with the **/test/sql** virtual path is used to connect to the database.

Parameters

You can pass parameters to the PL/SQL source file in the URL's query string as name-value pairs. In the PL/SQL code, you access these name-value pairs as "bind variables". For example, if you have the following URL:

```
http://machine.domain.name:port/test/sql/mysqlfile.sql?first_name=john
```

you can access the value of the `first_name` variable by prefixing it with a colon in the code:

```
declare
  n varchar2(32);
begin
  n := :first_name;          /* n is 'john' */
  http.htmlOpen;
  http.headOpen;
  http.title('Hello ' || n);
  http.headClose;
  http.bodyOpen;
  http.header(1, 'Sample for using .sql files');
  http.print('Hello ' || n);
  http.br;
  http.print('The URL is ' || owa_util.get_cgi_env('SCRIPT_NAME') ||
            owa_util.get_cgi_env('PATH_INFO') );
  http.br;
  http.print('This is from file' || owa_util.get_cgi_env('PATH_TRANSLATED') );
  http.bodyClose;
  http.htmlClose;
end;
```

Note that you cannot bind arrays in PL/SQL files because arrays are not supported as bind variables.

Life Cycle of the PL/SQL Cartridge

This section describes what the PL/SQL cartridge does when it receives a request. This section assumes knowledge of the callback functions used by the web request broker (WRB).

You do not need to know the information in this section in order to use the PL/SQL cartridge. However, this information is useful if you want to know the architecture of the cartridge.

Initialization

When a cartridge server process starts up, the `InitRuntime` callback function runs, and it initializes data structures used for all the cartridges in the application.

The `InitCartridge` callback function, which runs when the first instance of each cartridge starts up, reads the configuration information for the cartridge and sets up the database connection information.

The `InitInstance` callback function runs when each cartridge instance starts up, and it initializes request-specific data.

Authorization

The `Authorize` callback function is executed when a PL/SQL cartridge receives a request. The `Authorize` function:

- Checks if the requested object is protected under any authorization schemes or restrictions.
- Checks if the DAD contains username/password information is correct:
 - If not, the user is prompted to enter the username and password. The function logs into the database using the provided username and password.
 - If so, the function logs into the database and checks the authorization level set in the `owa_sec` package to determine if custom, or database-level, authentication is specified (See Chapter 5, “Authentication and Security” for details on custom authentication.). If so, the custom PL/SQL function that authenticates the user is executed.

Execution

If the `Authorize` callback function succeeds, the `Exec` callback function is called next. The `Exec` function:

- Gets the values of the CGI environment variables.

- Determines the PL/SQL procedure to run.
- Determines the parameters for the procedure.
- Builds PL/SQL scripts that bind the variables, and executes the scripts, which execute the procedure and write the output to the client through the WRB.

Shutdown

The Shutdown callback function is called automatically by the WRB. It closes all open connections.

Using the PL/SQL Web Toolkit

Contents

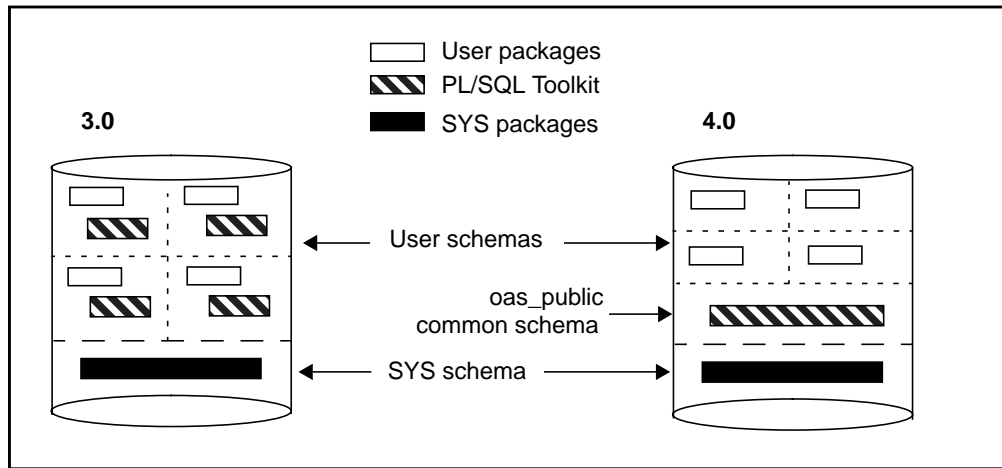
- [Common Schema](#)
- [PL/SQL Web Toolkit Installation](#)
- [Packages in the Toolkit](#)
- [Conventions for Parameter Names in the Toolkit](#)
- [Attributes to HTML Tags](#)
- [PL/SQL Cartridge and Applets](#)
- [Sessions/Cookies](#)
- [LONG Data Type](#)
- [Inter-Cartridge Exchange \(ICX\)](#)
- [Extensions to the http and htf Packages](#)
- [File Upload and Download](#)
- [String Matching and Manipulation](#)

Common Schema

Before you can use the PL/SQL cartridge, you must install the packages in the PL/SQL Web Toolkit in a common schema called “oas_public” in your Oracle database. You can install the Toolkit during the installation of the application server, or you can do it any time after you installed the application server using the Oracle Application Server administration forms.

The schemas where you load the packages have changed between versions 3.0 and 4.0. In 3.0 the packages had to be loaded in each schema accessed by PL/SQL cartridges. In 4.0 the packages are loaded in the “oas_public” common schema, and public synonyms are used to enable users to execute the objects in the common schema. Users execute the objects in the common schema with their own privileges, rather than with the privileges of the common schema.

Figure 4–1 Common schema for the PL/SQL Web Toolkit



If you are upgrading from 3.0, you need to remove the 3.0 packages from the individual schemas. The 4.0 packages will be installed automatically in a common schema.

PL/SQL Web Toolkit Installation

If you did not install the PL/SQL Web Toolkit when you installed Oracle Application Server, you can install it using Oracle Application Server Manager.

The installation script does the following:

- creates a user account for oas_public and installs the Toolkit in the schema.
 - creates a websys user and installs the Content service database objects and owa_content package in the websys schema.
1. Start up your browser and display the top-level administration page for Oracle Application Server.

2. Click OAS Utilities to display the Utilities navigational tree.
3. In the Utilities navigational tree, click the ☐ next to Install.
4. Under Install, click PLSQL Toolkit to display the Install form (Figure 4-2).
5. In the Install form, enter information on which Oracle database should contain the PL/SQL Web Toolkit:
 - ORACLE_SID: the name of the local Oracle database, if any.
 - Connect String: if the database is running on another machine, specify the SQL*Net V2 connect string here.
 - DBA Username and Password: the login to use to run the script that installs the PL/SQL Web Toolkit.

Note: You have to use SYS as the DBA user to install the Toolkit. Logging in as other users, including the SYSTEM user, will not work because they cannot grant execute privilege on the sys.dbms_sys_sql procedure to oas_public.

- Click Apply. This could take some time as the packages are being installed.

Figure 4-2 *PL/SQL Web Toolkit Install form*

The screenshot displays the PL/SQL Web Toolkit installation interface. On the left, a tree view under 'Utilities' shows 'Install' expanded, with 'PLSQL Toolkit' selected. The main area on the right is titled 'Install' and contains the following fields:

- ORACLE_SID :
- Connect String :
- DBA Username :
- Password :
- Confirm Password :

Packages in the Toolkit

The PL/SQL Web Toolkit contains the following packages:

Table 4–1 Packages in the PL/SQL Web Toolkit

Package	Description
htf and http	<p>The <code>http</code> (hypertext procedures) package contains procedures that generate HTML tags. For instance, the <code>http.anchor</code> procedure generates the HTML anchor tag, <code><A></code>.</p> <p>The <code>htf</code> (hypertext functions) package contains the function version of the procedures in the <code>http</code> package. The function versions do not directly generate output in your web page. Instead, they pass their output as return values to the statements that invoked them. Use these functions when you need to nest calls.</p> <p>To print the output of <code>htf</code> functions, call them from within the <code>http.print</code> procedure, which simply prints its parameter values to the generated web page.</p>
owa	Contains subprograms required by the PL/SQL cartridge.
owa_sec	Contains subprograms used by the cartridge for authenticating requests.
owa_util	<p>Contains utility subprograms. It is divided into the following areas:</p> <ul style="list-style-type: none">Dynamic SQL utilities enable you to produce pages with dynamically generated SQL code.HTML utilities enable you to retrieve the values of CGI environment variables and perform URL redirects.Date utilities enable correct date-handling. Date values are simple strings in HTML, but should be properly treated as a data type by the Oracle database.
owa_pattern	Contains subprograms that you can use to perform string matching and string manipulation with regular expression functionality.
owa_text	Contains subprograms used by <code>owa_pattern</code> for manipulating strings. They are externalized so you can use them directly.
owa_image	Contains subprograms that get the coordinates of where the user clicked on an image. Use this package when you have an imagemap whose destination links invoke a PL/SQL cartridge.

Table 4–1 Packages in the PL/SQL Web Toolkit

Package	Description
owa_cookie	Contains subprograms that enable you to send HTTP cookies to and get them from the client's browser. Cookies are opaque strings sent to the browser to maintain state between HTTP calls. State can be maintained throughout the client's session, or longer if an expiration date is included. Your system date is calculated with reference to the information specified in the owa_custom package.
owa_opt_lock	Contains subprograms that enable you to impose database optimistic locking strategies, so as to prevent lost updates. Lost updates can occur if a user selects and then attempts to update a row whose values have been changed in the meantime by another user.
owa_custom	Contains the authorize function (see “Custom Authentication” on page 5-4), and the time zone constants used by cookies.
owa_content	Contains subprograms that enable you to query the Content service repository and manipulate document properties. To use this package, the user needs execute privilege on this package. You can grant the privilege as the sys user.

http and htf

The `http` and `htf` packages provide subprograms that enable you to generate HTML tags from your stored procedure. For example, the following commands generate a simple HTML document:

```
create or replace procedure hello AS
BEGIN
    http.htmlopen;           -- generates <HTML>
    http.headopen;          -- generates <HEAD>
    http.title('Hello');    -- generates <TITLE>Hello</TITLE>
    http.headclose;         -- generates </HEAD>
    http.bodyopen;          -- generates <BODY>
    http.header(1, 'Hello'); -- generates <H1>Hello</H1>
    http.bodyclose;         -- generates </BODY>
    http.htmlclose;         -- generates </HTML>
END;
```

These packages also provide print procedures (such as [http.print](#)), which writes its argument to the current document. You can use these print procedures to generate non-standard HTML, to display the return value of functions, or to pass hard-coded text that appears in the HTML document as-is. The generated text is passed to the PL/SQL cartridge, which then sends it to the user's browser.

owa_image

The **owa_image** package contains subprograms that get the coordinates of where the user clicked on an image. You use this for imagemaps that invoke the PL/SQL cartridge. Your procedure would look something like:

```
create or replace procedure process_image
    (my_img in owa_image.point)
    x integer := owa_image.get_x(my_img);
    y integer := owa_image.get_y(my_img);
begin
    /* process the coordinate */
end
```

owa_opt_lock

The **owa_opt_lock** package contains subprograms that enable you to impose database optimistic locking strategies, so as to prevent lost updates. Lost updates can occur if a user selects and then attempts to update a row whose values have been changed in the meantime by another user.

The PL/SQL cartridge cannot use conventional database locking schemes because HTTP is a stateless protocol. The **owa_opt_lock** package works around this by giving you two ways of dealing with the lost update problem:

- The hidden fields method stores the previous values in hidden fields in the HTML page. When the user requests an update, the cartridge checks these values against the current state of the database. The update operation is performed only if the values match. To use this method, call the [owa_opt_lock.store_values procedure](#).
- The checksum method stores a checksum rather than the values themselves. To use this method, call the [owa_opt_lock.checksum function](#).

These methods are “optimistic”. That is, they do not prevent other users from performing updates, but they do reject the current update if an intervening update has occurred.

owa_custom

This feature is new to Oracle Application Server 4.0.

The **owa_custom** package contains the authorize function (see “[Custom Authentication](#)” on page 5-4), and the time zone constants used by cookies. Cookies use expiration dates defined in Greenwich Mean Time (GMT). If you are not on GMT, you can specify your time zone using one of these two constants:

- If your time zone is recognized by Oracle, you can specify it directly using `dbms_server_timezone`. The value for this is a string abbreviation for your time zone. (See *Oracle Server SQL Reference* under “SQL Functions” for a list of recognized time zones.) For example, if your time zone is Pacific Standard Time, you can use the following:

```
dbms_server_timezone constant varchar2(3) := 'PST'
```

- If your time zone is not recognized by Oracle, use `dbms_server_gmtdiff` to specify the offset of your time zone from GMT. Specify a positive number if your time zone is ahead of GMT, otherwise use a negative number.

```
dbms_server_gmtdiff constant number := NULL
```

After making the appropriate changes, you need to reload the package.

owa_content

The `owa_content` package contains functions and procedures that let you query the content service repository and manipulate document properties. You can use this package to perform tasks, like:

- set a document description
- delete documents
- delete document attributes
- retrieve attribute information
- list document attributes
- retrieve content type of a document

When compiling PL/SQL procedures and packages that use the `owa_content` package, you may get the following error message:

```
PLS-00201
```

```
identifier 'WEBSYS.OWA_CONTENT' must be declared
```

To avoid this error, when creating a new DAD that uses a non local database, you must enter the SYS username and corresponding password when prompted for a DBA user. Entering the SYSTEM user will not allow the correct grant and rights to be assigned to the database user. If you have entered SYSTEM as the DBA user then you must explicitly perform the grant privilege option as shown below:

```
SQL>grant all on WEBSYS.OWA_CONTENT to scott
```

If you are creating a DAD using an existing database user, you must perform the manual grant privilege shown above before using the OWA_CONTENT package.

The PL/SQL samples use the OWA_CONTENT package; so, these steps must be performed before installing the PL/SQL samples.

Conventions for Parameter Names in the Toolkit

In the PL/SQL Web Toolkit, the first letter of the parameter name indicates the data type of the parameter:

Table 4–2 *Parameter names in the PL/SQL Web Toolkit*

First character	Data type	Example
c	VARCHAR2	cname IN VARCHAR2
n	INTEGER	nsize IN INTEGER
d	DATE	dbuf IN DATE

Attributes to HTML Tags

Many HTML tags have a large number of optional attributes that, if passed as individual parameters to the hypertext procedures or functions, would make the calls cumbersome. In addition, some browsers support non-standard attributes. Therefore, each hypertext procedure or function that generates an HTML tag has as its last parameter *cattributes*, an optional parameter. This parameter enables you to pass the exact text of the desired HTML attributes to the PL/SQL procedure.

For example, the syntax for **htp.em** is:

```
htp.em(ctext, cattributes);  
A call that uses HTML 3.0 attributes might look like the following:  
  
htp.em('This is an example', 'ID="SGML_ID" LANG="en"');  
which would generate the following:  
  
<EM ID="SGML_ID" LANG="en">This is an example</EM>
```

PL/SQL Cartridge and Applets

When you reference an applet using the APPLET tag in an HTML file, the server looks for the applet class file in the directory containing the HTML file. If the applet class file is in another directory, you use the CODEBASE attribute of the APPLET tag to specify that directory.

When you generate an HTML page from the PL/SQL cartridge and the page references an applet, you must specify the CODEBASE attribute because the cartridge does not have a concept of a current directory and does not know where to look for the applet class file.

The following example uses [http.appletopen](#) to generate an APPLET tag. It uses the *attributes* parameter to specify the CODEBASE value.

```
http.appletopen('myapplet.class', 100, 200, 'CODEBASE="/applets"')  
generates
```

```
<APPLET CODE="myapplet.class" height=100 width=200 CODEBASE="/applets">
```

`/applets` is a virtual path that contains the **myapplet.class** file.

Sessions/Cookies

The Web Request Broker uses sessions to maintain persistent state within cartridges across multiple requests. Cookies can be used to maintain persistent state variables from the client browser. For information about cookies, see:

- http://home.netscape.com/newsref/std/cookie_spec.html
- <http://www.virtual.net/Projects/Cookies/>

The **owa_cookie** package enables you to send and retrieve cookies in HTTP headers. It contains the following subprograms that you can use to set and get cookie values:

- [owa_cookie.cookie data type](#) contains cookie name-value pairs.
- [owa_cookie.get function](#) gets the value of the specified cookie.
- [owa_cookie.get_all procedure](#) gets all cookie name-value pairs.
- [owa_cookie.remove procedure](#) removes the specified cookie.
- [owa_cookie.send procedure](#) generates a “Set-Cookie” line in the HTTP header.

Note: All HTTP headers have to be in English. If the headers are being generated from the database, make sure they are created in the English language.

LONG Data Type

If you use values of the LONG data type in procedures/functions such as [http.print](#), [http.prn](#), [http.prints](#), [http.ps](#), or [owa_util.cellsprint](#), be aware that only the first 64K of the LONG data is used. This reason for this limitation is that the LONG data is bound to a varchar2 data type in the procedure/function.

Inter-Cartridge Exchange (ICX)

If you are running Oracle database version 7.3.3 or later, you can call other cartridges from within your procedure using ICX. ICX enables a cartridge to communicate with other cartridges by making HTTP requests. The responses from the ICX calls can be received back by the calling cartridge (in this case, your procedure) for further processing.

To call cartridges from a stored procedure, you use the `utl_http` package. See the Oracle database documentation for details on the package. For the 7.3.3 release, the information is in the “readme” file. For the 8.x release, the information is in the *Application Developers Guide*.

Extensions to the http and htf Packages

The **http** and **htf** packages allow you to use customized extensions. Therefore, as the HTML standard changes, you can add new functionality similar to the hypertext procedure and function packages to reflect those changes.

Here is an example of customized packages using non-standard <BLINK> and imaginary <SHOUT>tags:

```
create package nsf as
    function blink(cbuf in varchar2) return varchar2;
    function shout(cbuf in varchar2) return varchar2;
end;

create package body nsf as
    function blink(cbuf in varchar2) return varchar2 is
        begin return ('<BLINK>' || cbuf || '</BLINK>');
    end;
    function shout(cbuf in varchar2) return varchar2 is
        begin return ('<SHOUT>' || cbuf || '</SHOUT>');
    end;
end;

create package nsp as
    procedure blink(cbufin varchar2);
```

```
procedure shout(cbufin varchar2);  
end;  
  
create package body nsp as  
  procedure blink(cbufin varchar2) is  
    begin http.print(nsf.blink(cbuf));  
  end;  
  procedure shout(cbufin varchar2) is  
    begin http.print(nsf.shout(cbuf));  
  end;  
end;
```

Now you can begin to use these procedures and functions in your own procedure.

```
create procedure nonstandard as  
begin  
  nsp.blink('Gee this hurts my eyes!');  
  http.print('And I might ' || nsf.shout('get mad!'));  
end;
```

File Upload and Download

The PL/SQL cartridge allows you to transfer files from a client machine to a database (uploading) and vice versa (downloading). You can upload and download text and binary files. The file upload/download feature of the PL/SQL cartridge is based on Oracle Application Server's Content Service.

Attributes of the uploaded files (such as file name, last modified date, content type, and owner) are also stored in the database. You can query the attributes and display only files that match the query criteria.

In the database, the files can be stored in uncompressed or compressed format, which is compatible with gzip.

Note: If you are using the listener that ships with Oracle Application Server, there is a limit of 8 million bytes that you can transfer.

Upload

To upload files from a client machine to a database, you create an HTML page that contains:

- A `FORM` tag whose `enctype` attribute is set to "multipart/form-data" and whose `action` attribute is associated with a PL/SQL cartridge function call.

- An INPUT element whose type and name attributes are set to file. The “INPUT type=file” element enables a user to browse and select files from the file system.

When a user clicks the submit button to trigger the form action, the following events occur:

1. The browser uploads the contents of the files specified by the user as well as other form data to the server.
2. The PL/SQL cartridge stores the file contents in the database.
3. The stored procedure specified in the action attribute is run similarly to invoking PL/SQL cartridges without file upload. The difference is that the file names are also passed to the procedure as an argument with other form data.

The DAD associated with the PL/SQL cartridge must be Content service-enabled. This means that the username associated with the DAD must be granted the ows_standard_role role in the database. If the DAD is not associated with a username, then the username that the user enters must be able to assume that role. This role enables the user to write the file contents to the Content service tables. Otherwise, an error occurs.

The following example shows an HTML form that enables a user to select a file from the file system to upload. The form contains other fields that allow the user to provide information about the file.

Figure 4–3 Upload form

The screenshot shows a Netscape browser window with the title "Netscape: test upload". The browser's menu bar includes File, Edit, View, Go, and Communicator. The address bar shows "http://myserver/upload.htm". The main content area displays a form titled "Upload Files". The form contains three text input fields: "Who:" with the value "john doe", "Description:" with the value "meeting minutes", and "File to upload:" with the value "/private/minutes.txt". A "Browse..." button is next to the "File to upload:" field. Below these fields is a "Submit Query" button. At the bottom of the form, it says "end of form" and "1 file uploaded files".

The HTML page is:

```
<html>
<head>
<title>test upload</title>
</head>

<body>
<p>start form
<FORM
    enctype="multipart/form-data"
    action="/sample/plsql/write_info"
    method="POST">
<p>Who:
<INPUT type="text" name="who">

<p>Description:
<INPUT type="text" name="description"><br>

<p>File to upload:
<INPUT type="file" name="file"><br>

<p>
<INPUT type="submit">

</FORM>
<p> end of form
</body>

</html>
```

When a user clicks the Submit button, the browser uploads the file listed in the “INPUT type=file” element. The write_info procedure is then run; the procedure writes information from the form fields to a table in the database, and returns a page to the user. It looks like this:

```
create procedure write_info (
    who          in varchar2,
    description in varchar2,
    file         in varchar2) as
begin
    insert into myTable values (who, description, file);
    http.htmlopen;
    http.headopen;
    http.title('File Uploaded');
```

```
http.headclose;  
http.bodyopen;  
http.header(1, 'Upload Status');  
http.print('Uploaded ' || file || ' successfully');  
http.bodyclose;  
http.htmlclose;  
  
end;
```

The procedure does not have to return anything to the user. But it is a good idea to let the user know whether the upload operation succeeded or failed.

In the stored procedure, you need to have an input parameter whose name is “file” because this is a required form element when uploading files. Its data type is either `varchar2` or `owa.vc_arr`. Use `varchar2` if you allow the user to upload a single file per submit action; use `owa.vc_arr` if you allow the user to upload multiple files per submit. To upload multiple files, you need to have multiple “INPUT type=file name=file” elements.

To store the uploaded file in compressed format, configure the cartridge to do so using the PL/SQL Parameters form in the Oracle Application Server Manager. In the **wrb.app** file, this corresponds to the `owa_compress_files` parameter. This parameter is set to `FALSE` by default.

Note: Depending on the platform, the browser can pass the file name without the full path to the server.

Download

After you have uploaded files to the database, you can download them, delete them from the database, and read and write their attributes.

To download files from a database to a user’s machine, you use the `http.download_file` procedure. The procedure has two variations:

```
http.download_file(sFileName in varchar2)  
http.download_file(sFileName in varchar2, bCompress in boolean)
```

The first parameter specifies the file to download, and the second parameter specifies whether the file should be compressed first before downloading.

After you download files from the database, do the following to see a list of downloaded files and their compression:

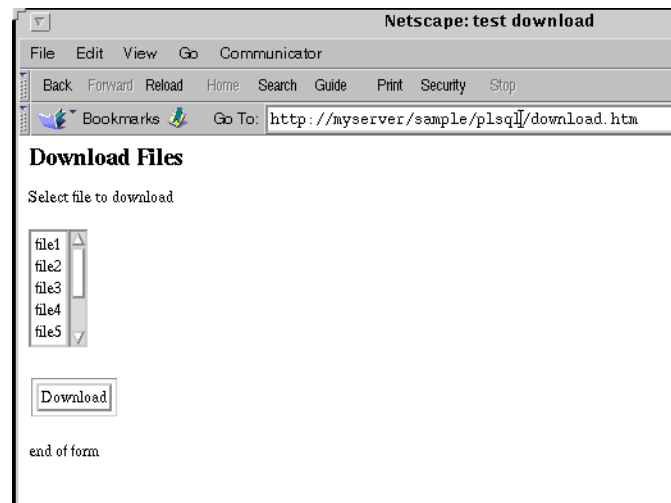
```
http.get_download_files_list(sFileName out varchar2)
```



```
http.get_download_files_list(bCompress out binary_integer);
```

The following example shows an HTML form that enables a user to download a file. The form displays a list of files available for download, and includes a submit button.

Figure 4–4 Download form



The HTML for the page is as follows:

```
<HTML>
<head>
<title>test download</title>
</head>

<body bgcolor="white">

<h2>Download Files</h2>

<FORM action="/sample/plsql/download" method="POST">

<p>Select file to download
<p>
<SELECT name="filex" size=5>
<option value="file1">file1</option>
```

```
<option value="file2">file2</option>
<option value="file3">file3</option>
<option value="file4">file4</option>
<option value="file5">file5</option>
<option value="file6">file6</option>
<option value="file7">file7</option>
<option value="file8">file8</option>
</select>

<p>
<INPUT type="submit" value="Download">

</FORM>
<p> end of form

<br>
<p>
</body>

</HTML>
```

The action associated with the download button invokes a stored procedure called `download` that downloads the selected file.

The code for the download procedure is:

```
create procedure download (filex in varchar2) as
begin
    http.download_file(filex);
end;
```

After the user clicks the download button, the browser prompts the user for the file-name under which to save the downloaded file. Note the following points about the procedure that downloads files:

- Calls to procedures that generate HTML, such as procedures in the HTP package, are ignored. It can invoke queries and stored procedures, but it cannot write strings that are returned to the browser. It can return only the contents of a file.
- The stored procedure can call `http.download_file` only once. You cannot download multiple files.
- If the downloaded files is compressed, use `gunzip` to uncompress the file.

Refer to Chapter 2, “The owa_content Package” for more utilities that enable you to manipulate the database file repository.

String Matching and Manipulation

The **owa_pattern** package contains procedures and functions that you can use to perform string matching and string manipulation with regular expression functionality. The package provides the following subprograms:

- The **owa_pattern.match function** determines whether a regular expression exists in a string. It returns TRUE or FALSE.
- The **owa_pattern.amatch function** is a more sophisticated variation of the **owa_pattern.match function**. It lets you specify where in the string the match has to occur. This function returns the end of the location in the string where the regular expression was found. If the regular expression is not found, it returns 0.
- The **owa_pattern.change function and procedure** lets you replace the portion of the string that matched the regular expression with a new string. If you call it as a function, it returns the number of times the regular expression was found and replaced.

These subprograms are overloaded. That is, there are several versions of each, distinguished by the parameters they take. Specifically, there are six versions of MATCH, and four each of AMATCH and CHANGE. The subprograms use the following parameters:

- *line* - This is the target to be examined for a match. Despite the name, it can be more than one line of text or can be a **owa_text.multi_line data type**.
- *pat* - This is the pattern that the subprograms attempt to locate in *line*. The pattern can contain regular expressions. Note in the **owa_pattern.change function and procedure**, this parameter is called *from_str*.
- *flags* - This specifies whether the search is case-sensitive or if substitutions are to be done globally.

owa_pattern.match

The regular expression in this function can be either a VARCHAR2 or a **owa_pattern.pattern data type**. You can create a **owa_pattern.pattern data type** from a string using the **owa_pattern.getpat procedure**.

You can create a `multi_line` data type from a long string using the [owa_text.stream2multi procedure](#). If a `multi_line` is used, the `rlist` parameter specifies a list of chunks where matches were found.

If the line is a string and not a `multi_line`, you can add an optional output parameter called `backrefs`. This parameter is a `row_list` that holds each string in the target that was matched by a sequence of tokens in the regular expression. Here is an example of the [owa_pattern.match function](#):

```
boolean foundMatch;  
foundMatch := owa_pattern.match('KAZOO', 'zoo.*', 'i');
```

This is how the function works: KAZOO is the target where it is searching for the “`zoo.*`” regular expression. The period indicates any character other than newline, and the asterisk matches 0 or more of the preceding characters. In this case, it matches any character other than the newline.

Therefore, this regular expression specifies that a matching target consists of “zoo,” followed by any set of characters neither ending in nor including a newline (which does not match the period). The *i* is a flag indicating that case is to be ignored in the search. In this case, the function returns TRUE, which indicates that a match had been found.

owa_pattern.change

[owa_pattern.change](#) can be a procedure or a function, depending on how it is invoked. As a function, it returns the number of changes made. If the flag ‘g’ is not used, this number can only be 0 or 1. The flag ‘g’ specifies that all matches are to be replaced by the regular expression. Otherwise, only the first match is replaced.

The replacement string can use the token ampersand (&), which indicates that the portion of the target that matched the regular expression is to be included in the expression that replaces it. For example:

```
owa_pattern.change('Cats in pajamas', 'C.+in', '& red ')
```

The regular expression matches the substring ‘Cats in’. It then replaces this string with ‘& red’. The ampersand character, &, indicates ‘Cats in’, since that’s what matched the regular expression. Thus, this procedure replaces the string ‘Cats in pajamas’ with ‘Cats in red’. If you called this as a function instead of a procedure, the value it would return would not be ‘Cats in red’ but 1, indicating that a single substitution had been made.

Authentication and Security

In addition to the authentication mechanisms provided by Oracle Application Server, the PL/SQL cartridge provides two additional levels of authentication mechanisms. The application server protects documents, virtual paths, and contents generated from the WRB, while the PL/SQL cartridge protects the users logging into the database or the PL/SQL web application itself.

For more information, see the security white paper at the application server developer's site (<http://technet.oracle.com>). This paper describes how to develop secure PL/SQL web applications.

Contents

- [Dynamic Username/Password Authentication](#)
- [Dynamic Username/Password and the Basic_Oracle Scheme](#)
- [PL/SQL Cartridge and Authentication Server Schemes](#)
- [Custom Authentication](#)

Dynamic Username/Password Authentication

In this scheme, access is controlled by the database itself; this scheme is suitable for an application that does not want to control the access on its own.

In the database access descriptor (DAD) form, enter valid username and password parameters values, but uncheck the Store Username and Password in the DAD checkbox. Thus, the users will be able to log into different schemas (database accounts) using the same PL/SQL cartridge/DAD combination. Users will be prompted with a dialog box in the browser to provide username and password information. This prompting happens during the authorization callback, and the

user-supplied information will be used to log into the database schema that belongs to the given username/password.

This scheme alleviates the problem of creating multiple PL/SQL cartridge/DAD combinations (DCDs in version 2.0) for multiple users, and it enables developers to write applications that access data from different schemas; the schemas correspond to the given username/password.

This is suitable for applications where multiple users with their own database accounts interact through the web applications. For example, for an intranet application serving 100 employees within that company, version 2.0 of the web server required 100 DCDs to be created with 100 different usernames and passwords, whereas in version 3.0 and later you just need to create one PL/SQL cartridge/DAD combination with no username and password.

Dynamic Username/Password and the Basic_Oracle Scheme

The `basic_oracle` scheme is a security scheme that enables you to validate users against an Oracle database. See the *Oracle Application Server Security Guide* for details.

Generally, you should use the `basic_oracle` scheme only for pages that do not use the dynamic username/password feature. In other words, you can use the `basic_oracle` scheme with PL/SQL cartridges when the DAD associated with the cartridge contains a username/password. The reason for this is that both methods use the database to authenticate the user and there could potentially be a conflict.

One reason that you might want to use the `basic_oracle` scheme in addition to dynamic username/password is if you need to validate users based on database roles. The `basic_oracle` scheme can authenticate users based on roles.

To do this, ensure that the `basic_oracle` scheme's username/password combination is going to be used for the PLSQL cartridge connection and also that scheme has all the necessary application code.

See the next section for information on how to specify PL/SQL cartridge virtual paths in security schemes. This is important because PL/SQL is case-insensitive while virtual paths are case-sensitive.

PL/SQL Cartridge and Authentication Server Schemes

You can use the authentication server schemes (such as basic, digest, IP-based restrictions, or domain-based restrictions) to protect the virtual paths that invoke the PL/SQL cartridges. The basic and digest schemes require the user to enter a

username and password, while the IP and domain schemes allow or restrict the user based on the user's IP address or domain name.

To associate virtual paths with schemes, use the Virtual Path page for each cartridge in the Oracle Application Server Manager. (You add usernames, group names, and realms to schemes using the Web Security pages in the Oracle Application Server Manager.)

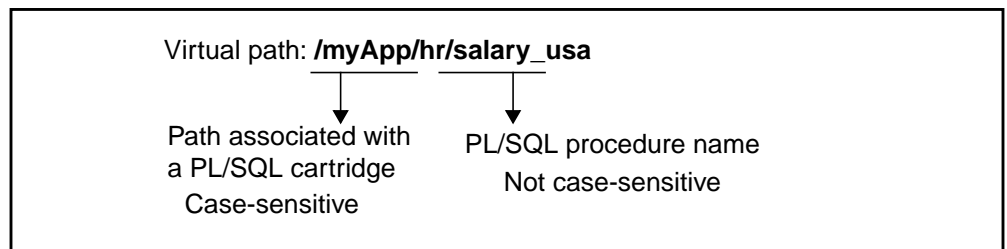
Because virtual paths are case-sensitive but PL/SQL procedure names are not, you should follow these guidelines when associating schemes with virtual paths for PL/SQL cartridges so that you do not expose any security risks:

- Protect all the stored procedures associated with a virtual path; do not protect just some stored procedures in a virtual path by qualifying the * wildcard character in the stored procedure component of the virtual path.

For example, if a PL/SQL cartridge is associated with the **/myApp/hr** virtual path, you should protect all the stored procedures by specifying **/myApp/hr/*** as the virtual path to protect. Do not specify something such as **/myApp/hr/salary*** to protect only procedures that begin with "salary". If you do, then when someone enters **/myApp/hr/SALARY_USA** as the virtual path, it would bypass authentication schemes associated with the virtual path because "SALARY" does not match "salary" in terms of case.

The portion of the virtual path up to, but not including, the stored procedure name is case-sensitive but the stored procedure name is not. This prevents users from entering **/MYAPP/HR/SALARY_USA** to run the SALARY_USA procedure because the dispatcher component of the application server does not match **/MYAPP/HR** with **/myApp/hr**.

Figure 5–1 Accounting for case when protecting virtual paths



- If you must protect only some procedures in a virtual path, use custom authentication with the PER_PACKAGE option. See the next section for details on custom authentication.

Custom Authentication

Custom authentication is suitable for applications that want to control the access themselves (that is, within the application itself). The application authenticates the users in its own level and not within the database level.

Custom authentication needs a static username/password to be stored in the configuration file, and cannot be combined with the dynamic username/password authentication.

A PL/SQL cartridge uses the username/password provided in the DAD to log into the database. Once the login is done, authentication control is passed to the application, and application-level PL/SQL hooks (callback functions) are called. The implementations for these callback functions are left to the application developers. The return value of the callback function determines if the authentication succeeded or failed: if the function returns TRUE, authentication succeeded. If it returns FALSE, authentication failed and code in the application is not executed.

The syntax of the `authorize` function is:

```
function authorize return boolean;
```

To enable custom authentication, you have to perform three steps:

1. Set the level of authentication by editing the **privcust.sql** file
2. Reload the application
3. Implement the authentication function.

If you enable custom authentication but do not define the callback function, you will get an error in the **wrb.log** file. Note that in the 3.0 release, the authentication level was done in the **privinit.sql** file, which does not exist in the 4.0 release.

You can place the authentication function in different locations, depending on when it is to be invoked:

- If you want the same authentication function to be invoked for all users and for all procedures, change the line in the **privcust.sql** file to:

```
owa_sec.set_authorization(OWA_SEC.GLOBAL)
```

and implement the **owa_custom.authorize** function in the “**oas_public**” schema, which contains the PL/SQL Web Toolkit.

- If you want a different authentication function to be invoked for each user and for all procedures, change the line in the **privcust.sql** file to:

```
owa_sec.set_authorization(OWA_SEC.CUSTOM)
```

and implement the **owa_custom.authorize** function in each user's schema. For users who do not have that function in their schema, the **owa_custom.authorize** function in the "oas_public" schema will be invoked instead.

For 3.0 users: if you implemented **owa_init.authorize** in each user's schema, you need to migrate the function to each user's **owa_custom** package.

- If you want the authentication function to be invoked for all users but only for procedures in a specific package or for anonymous procedures, change the line in the **prvcust.sql** file to:

```
owa_sec.set_authorization(OWA_SEC.PER_PACKAGE)
```

and implement the **authorize** function in that package in each user's schema. If the procedure is not in a package, then the anonymous **authorize** function is called instead. Table 5–1 summarizes the values for the parameters.

Table 5–1 Custom authorization

Value for parameter	Access control scope	Callback function
OWA_SEC.NO_CHECK	n/a	n/a
OWA_SEC.GLOBAL	All packages	owa_custom.authorize in the "oas_public" schema
OWA_SEC.PER_PACKAGE	Specified package	<i>packageName.authorize</i> in the user's schema
OWA_SEC.PER_PACKAGE	Anonymous procedures	authorize in the user's schema
OWA_SEC.CUSTOM	All packages	owa_custom.authorize in the user's schema, or, if not found, in the "oas_public" schema

When you use custom authentication, you can use the subprograms in the **owa_sec** package. You should not use **owa_sec** if you are not using custom authentication.

OWA_SEC.GLOBAL

This feature is new to 4.0.

The **owa_custom.authorize** function in the "oas_public" schema will be called whenever the PL/SQL cartridge is invoked.

For example, the following **authorize** function verifies that the user logged in as “guest” and specified “welcome” as the password and that the first and second numbers of the IP address of the client are 144 and 25.

```
create or replace package body owa_custom is
    -- Global authorize callback function
    -- It is used when the authorization scheme is set to OWA_SEC.GLOBAL

    function authorize return boolean is
        ip_address    owa_util.ip_address;
    begin
        -- prompt the user for login and password
        owa_sec.set_protection_realm('vendors');
        ip_address := owa_sec.get_client_ip;
        if ((owa_sec.get_user_id = 'guest') and
            (owa_sec.get_password = 'welcome') and
            (ip_address(1) = 144) and (ip_address(2) = 25)) then
            return TRUE;
        else
            return FALSE;
        end if;
    end;

begin -- OWA_CUSTOM
    owa_sec.set_authorization(OWA_SEC.GLOBAL);
end;
```

OWA_SEC.PER_PACKAGE

If the request specifies a procedure within a package, the **authorize** function in the package is invoked. If the procedure is not within a package, the anonymous **authorize** function is called.

For example, if the user invokes a procedure called **foo.print_page**, the **foo.authorize** function is called to authenticate the user.

```
create or replace package foo is
    procedure print_page;
    function authorize return boolean;
end;

create or replace package body foo is
    procedure print_page is
    begin
        http.print('Hello World');
    end;
```

```
function authorize return boolean is
begin
    owa_sec.set_protection_realm('vendors');
    if ((owa_sec.get_user_id = 'guest') and
        (owa_sec.get_password = 'welcome')) then
        return TRUE;
    else
        return FALSE;
    end if;
end; -- authorize function
end; -- package body foo

create or replace package body owa_custom is
    -- The authorize function in the owa_custom package will not
    -- be invoked if the authorization level is set at OWA_SEC.PER_PACKAGE.
begin -- OWA_CUSTOM
    owa_sec.set_authorization(OWA_SEC.PER_PACKAGE);
end;
```

OWA_SEC.CUSTOM

The **owa_custom.authorize** function in the user's schema will be called whenever the PL/SQL cartridge is invoked. This allows you to implement a different authorize function for each user. If the user's schema does not contain an **owa_custom.authorize** function, the PL/SQL cartridge looks for it in the "oas_public" schema.

Transactions

In versions of Oracle Application Server prior to 3.0, a procedure or set of procedures executed through a URI request committed all the changes/transactions done within that sequence of PL/SQL code. This may not be preferred for certain situations. For example, in electronic commerce applications, you want users to be able to add or remove items in their shopping basket without doing a commit every time the users make a change or invoke a new request. The preferred behavior is to show an updated view of the table with the new uncommitted row values, and allow the user to commit or abort the transaction.

Oracle Web Application Server 3.0 and Oracle Application Server 4.0 let you do that: the PL/SQL cartridge in this version supports the transaction service, which enables you to perform transactions that span several HTTP requests. The transaction service is based on the XA model transactions defined by the X/Open Company; the PL/SQL cartridge acts as a transactional model client and the database is used as the resource manager.

Chapter 10 “Enabling Transactions,” of the *Oracle Application Server Administration Guide* provides background and configuration information on transactions. You should read it before reading this chapter.

Contents

- [Mechanics of Transaction Service](#)
- [Example](#)

Mechanics of Transaction Service

Using the transaction service, you associate URIs with the operations on transactions (begin, commit, and rollback). When a user invokes one of these URIs, the corresponding transaction operation is performed by the transaction service.

These URIs can be mapped to either stored procedures or PL/SQL source files that display appropriate pages to the user. For example, the begin transaction URI could display to the user a list of items to add to his or her shopping cart, the commit transaction URI could display to the user a list of purchased items, and the rollback transaction URI could display to the user a page that asks if he wants to drop the existing shopping cart and start another one.

Between the begin and the commit or rollback URI, the user invokes other URIs that call procedures to perform some action on the database. These procedures might or might not belong to a transaction. If the URI belongs to a transaction, the actions performed by that procedure would be committed or rolled back when the transaction ends. If the URI does not belong to the transaction, it is not affected by the transaction, and Oracle Application Server treats it as a regular request (changes made by that URI are committed upon completion). URIs belonging to a transaction usually invoke procedures within a package.

The sequence of URIs for a PL/SQL stored procedures within the database when invoked would look like the following:

```
-- begin a transaction
http://host:port/myApp/cart1/test.txn_begin

-- the first operation in the transaction
http://host:port/myApp/cart1/test.txn_update1

-- the second operation in the transaction
http://host:port/myApp/cart1/test.txn_update2

-- some more operations

-- commit the transaction
http://host:port/myApp/cart1/test.txn_commit
```

In the example above, `test.begin`, `test.update1`, `test.update2`, and `test.commit` are procedures in the `test` package stored in the database. The `test` package marks the transaction boundary. You can give your procedures any name you like; the names used here are used only for clarification. If you are using PL/SQL source files, all of the names should have the suffix of “.sql”.

Note: The procedures that are associated with transactions, including the ones within a transaction, must **not** call the PL/SQL statements that commit or roll back transactions. If you do, you cannot use the transaction service model.

If an error occurs before the commit or rollback transaction, you need to roll back the transaction by calling the URI associated with the rollback transaction. Here is an outline of the `test.update1` procedure:

```
procedure test.update1 (...)
begin
    -- update some tables here

exception
    when appropriate_exception then
        owa_util.redirect_URI("/myApp/cart1/test.rollback");
end;
```

The [owa_util.redirect_url procedure](#) generates a `Location` header in the HTTP header. You cannot call the rollback transaction procedure directly from within other procedures.

Example

You could design an electronic commerce application that allows users to add items to their shopping carts, and the new values are not committed until the user clicks a Commit button. The example uses the values from the table above.

A transaction begins when the user invokes the URI:

```
http://host:port/myApp/cart1/txn_demo.begin_URI
```

The **txn_demo.begin_URI** procedure could generate an HTML page that displays to the user a list of items to add to his or her shopping cart. When the user adds an item to the shopping cart, the page would invoke a procedure that is within the transaction so that the addition is considered part of the transaction but is not committed (for example, **txn_demo.add_item**); the procedure that is invoked could just add a new row to a table in the database and generate a page that displays the contents of the user's shopping cart. The page would contain buttons that allow the user to commit or roll back the transaction. The commit button would invoke the **txn_demo.commit_URI** procedure, which could display to the user what he bought, and the rollback button would invoke the **txn_demo.rollback_URI** procedure, which could ask the user if he wants to start another shopping cart.

Contents

- [Supported Data Types](#)
- [NLS Extensions](#)
- [Upgrading PL/SQL Cartridge from 3.x to 4.0](#)

Supported Data Types

Because HTTP supports character streams only, the PL/SQL cartridge supports the following subset of PL/SQL data types.

- NUMBER
- VARCHAR2
- TABLE OF NUMBER
- TABLE OF VARCHAR2

Records are not supported.

NLS Extensions

The NLS extensions are part of the DAD configuration, and they provide a flexible infrastructure to request and retrieve values to and from Oracle databases in different languages/formats. Even when the database is configured with other NLS settings, all the conversions are handled implicitly by the database and the PL/SQL cartridge.

For example, if you have a database that is configured with US\$ for NLS Currency but you want to present the values in Japanese Yen to the user, all you need to do is

set NLS Currency to Japanese Yen. When the data is retrieved from the database, it will be presented as Japanese Yen.

The PL/SQL cartridge supports all the NLS extensions supported by the Oracle database. Versions of PL/SQL cartridge prior to Oracle Web Application Server 3.0 supported only the NLS_LANGUAGE parameter. This parameter was used by the cartridge to derive the NLS_LANGUAGE, NLS_TERRITORY, and NLS_CHARSET parameters.

Oracle Web Application Server 3.0 and Oracle Application Server 4.0 support these NLS extensions:

- NLS_DATE_FORMAT specifies the format to print dates in the client browser.
- NLS_DATE_LANGUAGE specifies the language to print day and month names in the client browser.
- NLS_SORT specifies the type of sort to use when sorting within the database.
- NLS_NUMERIC_CHARACTERS specifies the decimal character and the grouping separator character.
- NLS_CURRENCY specifies the local currency system to print monetary values in the client browser.
- NLS_ISO_CURRENCY specifies the ISO currency symbol.
- NLS_CALENDAR specifies the calendar system to print dates in the client browser.

The new NLS extension parameters are optional. If you do not provide values for these parameters, the default values are derived from the NLS_LANG parameter. For example, if the value of NLS_LANG is AMERICAN_AMERICA.US7ASCII:

- the values for NLS_DATE_LANGUAGE and NLS_SORT are derived from the language part of NLS_LANG, and
- the values for NLS_CURRENCY, NLS_DATE_FORMAT, NLS_ISO_CURRENCY, and NLS_NUMERIC_CHARACTERS are derived from the territory part of NLS_LANG

See the Oracle database documentation for details and valid values for these parameters.

Note: All the cartridges within an application should be configured to have the same NLS environment parameters. For example, NLS_LANG should be set only once for all cartridges within one single application.

Upgrading PL/SQL Cartridge from 3.x to 4.0

If you are upgrading from Oracle Web Application Server version 3.x to Oracle Application Server version 4.0, you have to perform the following steps in order to use the PL/SQL cartridge in 4.0:

- Follow the upgrade instructions in the “Upgrade” chapter in the *Oracle Application Server Installation Guide*. This step converts PL/SQL agents in 3.0 to PL/SQL cartridges in 4.0 and copies over the 3.0 DADs to the 4.0 environment.
- Check that you have installed the PL/SQL Web Toolkit in the “oas_public” common schema. You can install the toolkit when you install Oracle Application Server or at a later time using the Oracle Application Server Manager. See [“PL/SQL Web Toolkit Installation” on page 4-2](#) for details.

- Save any custom settings in the 3.0 owa_init package.

In the 3.0 toolkit, you could specify custom settings in the owa_init package. For example, you could specify the time zone and a custom authorize function.

If you want to save your 3.0 custom settings, copy them to the owa_custom package in 4.0. In 4.0, these settings are specified in the owa_custom package, not owa_init. Another option is to create a synonym for the owa_init package.

- Remove the 3.0 toolkit from individual schemas. Make sure that you have already saved your custom settings in the owa_init package, if any.
- Update your procedures, if necessary. The following procedures have been modified.
 - The [owa_util.mime_header procedure](#) now takes three parameters, instead of two.
 - The [owa_util.cellsprint procedure](#) has been overloaded to take an additional output parameter that specifies the number of rows that have been returned by the query.

Troubleshooting

Contents

- [Problems with Invoking Your PL/SQL Application](#)
- [Looking at Error Messages Generated by the Database](#)
- [Unhandled Exceptions](#)
- [Looking at the HTML Generated by Your PL/SQL Application](#)
- [Tracing Levels](#)
- [Error-Reporting Levels](#)

Problems with Invoking Your PL/SQL Application

If your PL/SQL application cannot be invoked:

- Make sure that the PL/SQL cartridge is registered with the WRB, and the virtual path for your application maps to the PL/SQL cartridge.
- Make sure that the listener and the WRB are functioning properly. For example, check that you can invoke other PL/SQL applications and other cartridges. You can try invoking the sample PL/SQL applications.
- Make sure that the PL/SQL subprogram that the URL references is a procedure, not a function.

Looking at Error Messages Generated by the Database

You can look at the database error messages that are returned to the user by setting the PL/SQL cartridge's error-reporting level to the highest value, which is 2. See "[Error-Reporting Levels](#)" for details on the different error levels.

If you have logging turned on for the PL/SQL cartridge, database messages are logged to the specified log file (**wrb.log**). To turn logging on for the cartridge, go to the Application Configuration/Logging page, and check that the PL/SQL cartridge has logging enabled. Also check the logging level; by setting the logging level to a higher level, you can get more messages. See [Tracing Levels](#) for details.

Unhandled Exceptions

If an error occurs in your PL/SQL procedure, an exception is thrown. If you do not handle the exception, the error is logged in the log file with the Oracle error stack and an error message is returned to the user. The error-reporting level controls what the user sees. See [Error-Reporting Levels](#) for details on the different error levels.

Recall that when a procedure exits with an unhandled exception, PL/SQL does not assign values to OUT parameters and does not commit database work done by your procedure.

You can avoid unhandled exceptions by coding an OTHERS handler at the top level of your procedure.

Looking at the HTML Generated by Your PL/SQL Application

The PL/SQL Web Toolkit provides the [owa_util.showpage procedure](#), which you can use in Oracle Server Manager to print out the output of your application. The following example prints out the HTML generated by the **current_users** procedure (which was shown in the [Tutorial](#) section).

```
% svrmgr1
Oracle Server Manager Release 2.3.2.0.0 - Production
Copyright (c) Oracle Corporation 1994, 1995. All rights reserved.
Oracle7 Server Release 7.3.2.1.0 - Production Release
With the distributed option
PL/SQL Release 2.3.2.0.0 - Production
SVRMGR> connect scott/tiger
Connected.
SVRMGR> set serveroutput on
Server Output      ON
SVRMGR> execute current_users
Statement processed.
SVRMGR> execute owa_util.showpage
Statement processed.
<HTML>
<HEAD>
```



```
<TITLE>Current Users</TITLE>
</HEAD>
<BODY>
<H1>Current Users</H1>
<TABLE >
<TR>
<TH>USERNAME</TH>
<TH>USER_ID</TH>
<TH>CREATED</TH>
</TR>
<TR>
<TD ALIGN="LEFT">SYS</TD>
<TD ALIGN="LEFT">0</TD>
<TD ALIGN="LEFT">21-JAN-97</TD>
</TR>
<TR>
<TD ALIGN="LEFT">SYSTEM</TD>
<TD ALIGN="LEFT">5</TD>
<TD ALIGN="LEFT">21-JAN-97</TD>
</TR>
<TR>
<TD ALIGN="LEFT">WWW_USER</TD>
<TD ALIGN="LEFT">11</TD>
<TD ALIGN="LEFT">27-JAN-97</TD>
</TR>
<TR>
<TD ALIGN="LEFT">TRACESVR</TD>
<TD ALIGN="LEFT">8</TD>
<TD ALIGN="LEFT">21-JAN-97</TD>
</TR>
<TR>
<TD ALIGN="LEFT">SCOTT</TD>
<TD ALIGN="LEFT">9</TD>
<TD ALIGN="LEFT">21-JAN-97</TD>
</TR>
<TR>
<TD ALIGN="LEFT">WWW_DBA</TD>
<TD ALIGN="LEFT">10</TD>
<TD ALIGN="LEFT">27-JAN-97</TD>
</TR>
</TABLE>
</BODY>
</HTML>
```

Tracing Levels

You can get detailed information about what the PL/SQL cartridge is doing by increasing the tracing level. The tracing messages are printed only to the **wrb.log** file; they are not sent to the user.

The severity levels range from 0 to 15; low values indicate that only errors are logged, while high values indicate that warnings and informative messages are also logged. For example, if you set the severity level to 8, you can see when the cartridge has performed the authentication and execution operations. The following table describes the severity levels:

Table 8–1 Severity levels

Meaning	Severity	Recommended usage
Fatal errors (for example, memory errors)	0	0 when a core failure occurred.
	1	1 when writing to file or database failed.
Soft errors (for example, non-fatal input/output errors)	2	(user-defined)
	3	(user-defined)
Warnings (for example, missing file or missing configuration section)	4	4 when a configuration error occurred, such as a file or directory does not exist, or a section in a configuration file is missing.
	5	(user-defined)
	6	(user-defined)
Tracings (for example, request has been executed)	7	7 to trace process events, for example, a process's initialization, reload, and termination stages.
	8	8 to trace thread events, for example, a thread's initialization and termination stages.
	9	9 to trace request events, for example, a request has been received.
	10	10 for messages that occur while executing a request.
	11	11 for messages that occur while authorizing a request.
	12	(user-defined)

Table 8–1 Severity levels

Meaning	Severity	Recommended usage
Debugging (for example, variable logging)	13	13 is used for printing debugging variables.
	14	(user-defined)
	15	(user-defined)

Error-Reporting Levels

The PL/SQL cartridge supports three levels of error-reporting. These levels control what the user sees when an error occurs. In versions older than 3.0, the PL/SQL cartridge displayed a static file in case of errors, and it was not possible to identify the error from the browser.

The error levels are configured as part of the PL/SQL cartridge. Errors are reported only during the Exec callback function.

Table 8–2 Error-reporting levels

Error level	Description
0	<p>Displays a static file in the client browser when an error occurs. Use this level if you do not want the users to see any information about the error.</p> <p>You can specify the file to return to the client. The default file is \$ORAWEB_HOME/./cartx/plsql/install/error.html.</p>
1	<p>Displays the date and time of the error, and also the URL that caused the error. Use this level if you want to provide only minimal information for the user to pass it on to web site managers or application developers. The site manager or developer can use this information to help diagnose the error in the log file.</p> <p>If this error level is specified, the error page (if specified) is ignored.</p> <p>Example:</p> <p><i>Error occurred while accessing “/test/myproc” at Mon Jan 6 16:33:32 1997.</i></p>

Table 8–2 Error-reporting levels

Error level	Description
2	<p>Displays detailed information such as date and time of the error, the URL, the cartridge name, the procedure that was called, the parameter names and values, the web server error code, and the database error with a call stack. This error level is typically used only while developing or debugging an application.</p> <p>Example:</p> <pre>Error occurred at Mon Jan 6 16:33:32 1997 OWS-05101: Agent: exexution failed due to Oracle error 6564 ORA-06564: object show_stats does not exist ORA-06512: at "SYS.DBMS_DESCRIBE", line 55 ORA-06512: at line 1 OWA SERVICE: OWA_DEFAULT_SERVICE PROCEDURE: show_stats</pre>

Part II

ODBC Cartridge

ODBC Cartridge Overview

The ODBC cartridge uses Open Database Connectivity, an interface enabling applications to access data in database management systems that use Structured Query Language (SQL). The ODBC standard defines a vendor-independent interface for accessing data.

Contents

- [Review of Cartridge Architecture](#)
- [Supported Data Sources](#)

Review of Cartridge Architecture

The Oracle ODBC cartridge is integrated with the Oracle Application Server, creating a bridge to Oracle and other ODBC databases. The ODBC cartridge uses an ODBC driver manager and ODBC drivers specific to each database. The ODBC cartridge architecture has four components:

- the ODBC cartridge, a database application that makes the connection to the ODBC driver
- a driver manager that implements the ODBC application programming interface, loads drivers, and provides argument checking and state-transition checking
- the drivers that process ODBC function calls and manage exchanges between the ODBC cartridge and a data source
- a data source that contains the information the ODBC cartridge needs about the database management system, its platform, and the network to be used

Supported Data Sources

The ODBC cartridge supports these data sources:

- Oracle Server
- Sybase SQL Server Systems
- INFORMIX-OnLine Dynamic Server database server
- INFORMIX-SE database server
- Microsoft SQL Server

For a list of current versions supported go to the Website **<http://technet.oracle.com>**.

Using the ODBC Cartridge

Contents

- [Invoking the ODBC Cartridge](#)
- [Special Usage Considerations](#)

Invoking the ODBC Cartridge

To invoke the ODBC cartridge, send an HTTP URL request to the Oracle Application Server using this syntax:

`http://hostname:port/odbc/request_mode`

where:

- *hostname:port* identifies the Oracle Application Server machine.
- *odbc* identifies the virtual path to the ODBC cartridge.
- *request_mode* identifies the type of request to the cartridge. Valid values are `execute`, `tableprint`, or `stringprint`. These request modes include parameters specific to your database, such as data source name. See [“Specifying a Request Mode” on page 11-1](#).

Special Usage Considerations

1. If a SQL statement refers to a parameter and the parameter is not passed, no default value is used and you receive an error message.
2. NULL is a reserved keyword and cannot be used as a string value.
3. If a SQL statement refers to a parameter with a value that contains only blanks, NULL is used as the value of that parameter.

Specifying Modes and Datatypes

To use the cartridge, you invoke the cartridge by sending an HTTP URL request. To do this, you need to know the structure to use for your request and how different datatypes are supported

Contents

- [Specifying a Request Mode](#)
- [Specifying Datatypes](#)

Specifying a Request Mode

When you invoke the ODBC cartridge you specify one of three request modes:

- [Execute Mode](#) executes the SQL statement you supply but returns no data
- [TablePrint Mode](#) produces the results of the SQL in an HTML table
- [StringPrint Mode](#) produces the result of the SQL as a series of strings

Request Mode Parameters

The request modes use parameters specific to your database. Each type of request mode uses some combination of the following parameters:

<code>dsn</code>	the data source name that you chose while configuring the ODBC environment.
<code>username</code>	is optional, and provides the username to log onto the database
<code>password</code>	is optional, and contains the database user's password to log onto the database

`sql` contains the ODBC SQL string modified to URL form, and includes input parameters.

Input parameters identify the location where a value is to be placed in the SQL statement. To specify input parameters, precede the parameter name with a colon (:) and supply a value corresponding to this name.

Use this syntax to specify the value for an input parameter in the URL request:

input_parameter_name[_suf]=input_parameter_value

where *input_parameter_name* is the input parameter name and *[_suf]* is the optional suffix for specifying the datatype explicitly. *input_parameter_value* uses a format based on the datatype categories:

- **char** = xxxxxxxx (depends on the length)
- **integer** = nnnnn (number of digits depends on the precision)
- **decimal** = nnnn.nnnn (depends on precision and scale)
- **timestamp** = yyyy-mm-dd hh:mm:ss.[fff]
- **binary** = hexadecimal string in which two characters represent a byte

See [“Specifying Datatypes” on page 11-4](#) for information about using suffixes to specify the datatype explicitly.

In this example of a SQL statement in a URL request, the **:name** input parameter has a corresponding value of Scott Champion, the **:age** input parameter has a corresponding value of 26, and the **:department** input parameter has a corresponding value of IT:

```
http://mark1:8888/odbc/execute?dsn=sybl0&username=scott&password=tiger
&sql=insert+into+emp+values(:name,:age,:department)
&name=Scott+Champion&age=26&department=IT
```

`outputstring` specifies the **printf** style used to output the results. Specify the mapping from row data into the string using the syntax **%n** where **n** is the column number to be inserted.

`maxrows` is optional and indicates the maximum number of rows to display. The default is 25 rows.

`minrows` is optional and indicates the minimum number of rows to display. The default is 0.

Execute Mode

In Execute mode the SQL statement you supply is executed, but no data is returned. You receive only notification of the request's success or failure. Use this mode for DDL operations and DML inserts and updates.

The parameters for Execute mode are:

- `dsn`
- `username`
- `password`
- `sql`

Here is an example of a URL invoking the ODBC cartridge to connect to an Oracle7 database using Execute mode:

```
http://mark1:8888/odbc/execute?dsn=orcl7&username=scott
&password=tiger&sql=insert+into+emp+values(:name,:age,:dept)
&name=Hamilton&age=26&dept=IT
```

TablePrint Mode

TablePrint mode produces the results of the SQL statement in an HTML table. Valid parameters are:

- `dsn`
- `username`
- `password`
- `sql`
- `maxrows`
- `minrows`

Here is an example of a URL invoking the ODBC cartridge using TablePrint mode:

```
http://mark1:8888/odbc/tableprint?dsn=orcl7&username=scott&password=tiger
&sql=select+*+from+emp+where+deptno=:department
&department=IT&maxrows=7
```

StringPrint Mode

StringPrint mode produces the result of the SQL statement as a series of strings, as if the values for each row were parameters to a **printf** string supplied by the requester. Valid parameters are:

- `dsn`
- `username`
- `password`
- `sql`
- `outputstring`
- `maxrows`

Here is an example of a URL invoking the ODBC cartridge using StringPrint mode:

```
http://mark1:8888/odbc/stringprint?dsn=orcl7&username=scott&password=tiger
&sql=select+name,age+from+emp+where+deptno=:department
&department=IT&outputstring=Employee+%1+is+%2+years+old.&maxrows=7
```

Specifying Datatypes

Different databases use different SQL datatypes. When specifying the value for an input parameter in the `sql` portion of a URL request, you can use a suffix to specify a specific datatype for the parameter.

If you do not specify a suffix, the ODBC cartridge accepts the input parameters as string literals and sends them to the ODBC driver as such. The cartridge relies on the ability of your ODBC driver and the target database to handle implicit conversions of the string literals to the appropriate datatype. If the ODBC driver or target database cannot handle the implicit datatype conversion, an error condition results. To ensure datatypes are converted correctly, specify the datatype of your input parameter values using the datatype suffix.

This example uses suffixes to specify the datatypes explicitly for each input parameter in the URL request. In this example, `name_char` specifies the CHAR datatype, `age_intg` specifies the INTEGER datatype, and `dept_char` specifies the CHAR datatype:

```
http://mark1:8888/odbc/
execute?sql=insert+into+emp+values(:name_char,:age_intg,:dept_char)
&name_char=Scott&age_intg=30&dept_char=IT
```

Look up the datatype information pertaining to your target database:

- [Specifying Oracle datatypes](#)
- [Specifying Sybase datatypes](#)
- [Specifying Informix datatypes](#)
- [Specifying Microsoft SQL Server datatypes](#)

Table 11–1 Specifying Oracle datatypes

Datatype	Suffix	Comments
CHAR	_CHAR	Character string of fixed-string length from 1 to 255
DATE	_DTME	Date/time data: <i>yyyy-mm-dd hh:mm:ss</i> format
FLOAT	_FLOT	no comments
LONG	_LCHR	no comments
LONG RAW	_LBIN	no comments
NUMBER	_FLOT	no comments
NUMBER(38)	_INTG	no comments
NUMBER(P)	_DECL	Signed, exact, numeric value with a precision P
NUMBER(P,S)	_DECL	Signed, exact, numeric value with a precision P and scale S
RAW	_BINY	See restriction (Unix only) no comments (NT only)
VARCHAR2	_VCHR	no comments

Table 11–2 Specifying Sybase datatypes

Datatype	Suffix	Comments
BINARY	_BINY	Binary data of fixed length from 1 to 255
BIT	_BBIT	Single bit binary data with the value of 0 or 1
CHAR	_CHAR	Character string of fixed-string length from 1 to 255
DATETIME	_DTME	Date/time data: <i>yyyy-mm-dd hh:mm:ss.fff</i> , <i>yyyy-mm-dd hh:mm:ss</i> , or <i>yyyy-mm-dd</i> format

Table 11–2 Specifying Sybase datatypes

Datatype	Suffix	Comments
DECIMAL(P S)	_DECL	Signed, exact, numeric value with a precision P and scale S
FLOAT	_FLOT	no comments
IMAGE	_LBIN	no comments
INT	_INTG	Range is from $(2^{**}31)-1$ to $-2^{**}31$
MONEY	_DECL	no comments
NCHAR	_CHAR	Character string of fixed-string length from 1 to 255
NUMERIC(P,S)	_NUMR	Signed, exact, numeric value with a precision P and scale S
NVARCHAR	_VCHR	Variable-length character string with a string length of from 1 to 255 characters
REAL	_REAL	no comments
SMALLDATETIME	_DTME	Date/time data
SMALLINT	_SINT	Range is from -32768 to 32767
SMALLMONEY	_DECL	no comments
SYSNAME	_VCHR	no comments
TEXT	_LCHR	no comments
TIMESTAMP	_VBIN	no comments
TINYINT	_TINT	Range is from 0 to 255
VARBINARY	_VBIN	no comments
VARCHAR	_VCHR	no comments

Table 11–3 Specifying Informix datatypes

Datatype	Suffix	Comments
BYTE	_LBIN	Datatype used only by INFORMIX-OnLine
CHAR	_CHAR	no comments
DATE	_DATE	Date data: <i>yyyy-mm-dd</i> format
DATETIME	_DTME	Date/time data: <i>yyyy-mm-dd hh:mm:ss:fff</i> format

Table 11–3 Specifying Informix datatypes

Datatype	Suffix	Comments
DECIMAL(P,S)	_DECL	no comments
DECIMAL(P)	_DUBL	no comments
FLOAT	_DUBL	no comments
INTEGER	_INTG	no comments
INTERVAL	_CHAR	no comments
MONEY	_DECL	no comments
SERIAL	_INTG	no comments
SMALLFLOAT	_REAL	no comments
SMALLINT	_SINT	Range is from -32768 to 32767
TEXT	_LCHR	Datatype used only by INFORMIX-OnLine
VARCHAR	_VCHR	Datatype used only by INFORMIX-OnLine

Table 11–4 Specifying Microsoft SQL Server datatypes

Datatype	Suffix	Comments
BINARY	_BINY	Binary data of fixed length from 1 to 255
BIT	_BBIT	Single bit binary data
CHAR	_CHAR	Character string of fixed-string length from 1 to 255
DATETIME	_DTME	Date/time data: yyyy-mm-dd hh:mm:ss.fff, yyyy-mm-dd hh:mm:ss, or yyyy-mm-dd format
DECIMAL	_DECL	no comments
FLOAT	_FLOT	no comments
IMAGE	_LBIN	no comments
INT	_INTG	no comments
MONEY	_DECL	no comments
NUMERIC	_NUMR	no comments
REAL	_REAL	no comments
SMALLDATETIME	_DTME	no comments

Table 11–4 Specifying Microsoft SQL Server datatypes

Datatype	Suffix	Comments
SMALLINT	_SINT	Range is from -32768 to 32767
SMALLMONEY	_DECL	no comments
SYSNAME	_VCHR	no comments
TEXT	_LCHR	no comments
TIMESTAMP	_VBIN	no comments
TINYINT	_TINT	no comments
VARBINARY	_VBIN	no comments
VARCHAR	_VCHR	no comments

Index

A

- applications
 - PL/SQL cartridges, adding, 3-2
- architecture
 - ODBC cartridge, 9-1
- authentication
 - PL/SQL cartridge, 5-1, 5-4
- authentication server security schemes, 5-2
- authorization, 3-20
- authorization, custom
 - PL/SQL cartridge, 5-5
- authorize function
 - PL/SQL cartridge, 5-4

B

- basic_oracle security scheme
 - PL/SQL cartridge, 5-2

C

- callback functions
 - authentication functions, 5-4
- cookies
 - PL/SQL cartridge, 4-5, 4-9
 - PL/SQL Web Toolkit, 4-9
- custom authentication
 - enabling, 5-4
 - PL/SQL cartridge, 5-4
 - values, 5-5
- custom authorization
 - PL/SQL cartridge, 5-5

D

- DAD. *See* Database Access Descriptor (DAD)
- data types supported in PL/SQL cartridge, 7-1
- Database Access Descriptor (DAD), 1-2
 - creating, 2-3, 3-3
 - dialog box, 2-4
- datatypes
 - specifying, 11-4
- differences in 4.0
 - transaction service, 6-1
- Download form, 4-15
- dynamic username/password security scheme
 - PL/SQL cartridge, 5-2

E

- error messages
 - generated by database, 8-1
- error-reporting levels in the PL/SQL cartridge, 8-5
- exceptions, unhandled, 8-2
- exec function
 - errors reported, 8-5
- Execute, 11-3
- execute mode, 11-3
- execution, 3-20

F

- file privcust.sql, 5-4
- foo.authorize function, 5-6
- functions
 - authorize, 5-4
 - foo.authorize, 5-6

owa_custom.authorize, 5-4

H

HTML

- extending, 4-10
 - generated by PL/SQL application, 8-2
- HTML elements
- attributes, 4-8

I

ICX service

- PL/SQL cartridge, 4-10

Informix datatypes, 11-6

initialization, 3-20

L

listener port numbers

- and PL/SQL agents, 3-10

LONG data type, 4-10

M

Microsoft SQL server datatypes, 11-7

N

NLS

- extensions in PL/SQL cartridge, 7-1

NLS extensions, 7-1

O

oas_public schema, 4-2

ODBC cartridge

- invoking, 11-1
- request mode, 11-1
- specifying datatypes, 11-4

Oracle database files, 2-2

Oracle datatypes, 11-5

ORACLE_HOME directory, 2-2

ORACLE_SID field, 2-4

overloaded procedures

- PL/SQL cartridge, 3-10

overview

- PL/SQL cartridge, 1-1

owa_custom.authorize function, 5-4

owa_pattern.change function or procedure, 4-18

owa_pattern.match function, 4-17

OWA_SEC.CUSTOM variable, 5-7

OWA_SEC.GLOBAL variable, 5-5

OWA_SEC.PER_PACKAGE variable, 5-6

owa_sec.set_authorization procedure, 5-4

owa_util.showpage procedure, 8-2

owains.sql, 4-2

P

packages

htf package, 4-4

htp package, 4-4

PL/SQL cartridge

- extending htp and htf packages, 4-10

installing, 2-3

owa package, 4-4

owa_content package, 4-5

owa_cookie package, 4-5

owa_custom package, 4-5

owa_image package, 4-4, 4-6

owa_init package, 7-3

owa_opt_lock package, 4-5

owa_pattern, 4-4

owa_sec package, 4-4

owa_text package, 4-4

owa_util package, 4-4

parameters

- NLS extensions, 7-2

physical path

- for PL/SQL cartridge, 3-3

- for PL/SQL source files, 3-19

PL/SQL agents

- and listener port numbers, 3-10

PL/SQL application

adding, 3-1

adding and invoking, 3-1

configuring, 3-7

HTML generated, 8-2

PL/SQL arrays

- and overloading, 3-11

- PL/SQL cartridge
 - adding applications and cartridges, 3-2
 - and ICX, 4-10
 - authentication and security, 5-1
 - cookies, 4-9
 - creating a DAD, 2-3
 - custom authentication, 5-4
 - DAD, 1-2
 - data types supported, 7-1
 - dynamic username/password
 - authentication, 5-1
 - error-reporting levels, 8-5
 - executing SQL files, 3-18
 - installing packages, 2-3
 - invoking, 3-1, 3-8
 - life cycle, 3-20
 - LONG data type, and, 4-10
 - NLS extensions, 7-1
 - overloaded procedures, 3-10
 - overview, 1-1
 - packages
 - extending http and htf packages, 4-10
 - htf package, 4-4
 - http, 4-4
 - owa, 4-4
 - owa_content, 4-5
 - owa_cookie, 4-5
 - owa_custom, 4-5
 - owa_image, 4-4, 4-6
 - owa_init, 7-3
 - owa_opt_lock, 4-5
 - owa_pattern, 4-4
 - owa_sec, 4-4, 5-5
 - owa_text, 4-4
 - owa_util, 4-4
 - parameters
 - passing to PL/SQL source files, 3-19
 - positional, 3-17
 - parameters passed to subprograms, 4-8
 - request processing, 1-3
 - severity levels, 8-4
 - string matching and manipulation, 4-17
 - tracing levels, 8-4
 - transactions, 6-1
 - example, 6-3

- troubleshooting, 8-1
- tutorial, 2-1
- upgrading, 7-3
- URL to invoke the PL/SQL cartridge, 1-1
- username/password authentication, 5-1
- using the PL/SQL Web Toolkit, 4-1
- variables with multiple values, 3-12
- PL/SQL Parameters form, 3-8
- PL/SQL table in PL/SQL cartridge, 3-12
- PL/SQL Web Toolkit
 - attributes to HTML tags, 4-8
 - common schema, 4-1
 - customizing, 4-10
 - extensions to packages, 4-10
 - ICX, 4-10
 - Install form, 4-3
 - installing, 4-2
 - packages, 4-4
 - parameter names, 4-8
 - PL/SQL cartridge and applet, 4-8
 - sessions/cookies, 4-9
 - string matching and manipulation, 4-17
- privcust.sql file, 5-4
- protection
 - in PL/SQL cartridge, 5-1

R

- request mode parameters, 11-1

S

- security
 - authentication server schemes, 5-2
 - in PL/SQL cartridge, 5-1
- sessions
 - PL/SQL Web Toolkit, 4-9
- severity levels
 - PL/SQL cartridge, 8-4
- shutdown, 3-21
- specifying datatypes, 11-4
- SQL files
 - executing, 3-18
- stored procedures
 - creating, 2-2

- invoking with URLs, 6-1
- stringprint mode, 11-4
- strings
 - matching and manipulating in the PL/SQL cartridge, 4-17
- Sybase datatypes, 11-5

T

- tableprint mode, 11-3
- tracing levels
 - PL/SQL cartridge, 8-4
- transaction service
 - PL/SQL cartridge, 6-1
 - example, 6-3
- troubleshooting
 - PL/SQL cartridge, 8-1
- tutorial
 - PL/SQL cartridge, 2-1

U

- unhandled exceptions, 8-2
- upgrading
 - PL/SQL cartridge, 7-3
- Upload form, 4-12
- URLs
 - transaction service, 6-1

V

- virtual path
 - for PL/SQL cartridge, 3-3
 - protecting, 5-3